VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Jan Broda

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

APLIKACE PRO PROGRAMOVATELNÉ ČIPOVÉ KARTY

APPLICATION FOR PROGRAMMABLE SMART CARDS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

Jan Broda

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. Jan Hajný, Ph.D.

BRNO 2020

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
TECHNICKÉ V BRNĚ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Jan Broda ***ID:*** 203697
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Aplikace pro programovatelné čipové karty

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je návrh a vývoj aplikace pro silnou autentizaci na platformě čipových karet. Student v rámci bakalářské práce naprogramuje aplikaci pro čipovou kartu v jazyce C, která bude realizovat schéma atributové autentizace CDDH19 s revokačním schématem CDH16. Výstupem práce je funkční aplikace pro čipovou kartu Multos.

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] MULTOS Developer's Guide [online]. , 96 [cit. 2019-09-06]. Dostupné z: https://www.multos.com/uploads/MDG.pdf

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 bakalářská práce se v teoretické části zabývá problematikou bezpečné atributové autentizace, která klade vysoký důraz na ochranu soukromí a digitální identity. Dále je v teoretické části pojednáváno o čipových kartách s operačním systémem MultOS. Praktickým cílem práce je vytvoření autentizační aplikace v jazyce C s využitím atributového schématu CDDH19 a revokačního schématu CDH16. Atributové schéma je založeno na podpisovém schématu weak Boneh Boyen a algebraické MAC funkci. Pro vývoj bylo využito rozhraní Eclipse IDE s vývojovým prostředím SmartDeck. Výsledná aplikace uživateli umožní bezpečnou atributovou autentizaci bez rizika úniku citlivých informací. Aplikace je implementována na MultOS čipovou kartu a otestována ve spolupráci se studentem, který naprogramoval stranu ověřovatele na mikropočítači Raspberry PI.

KLÍČOVÁ SLOVA

atribut, autentizace, MultOS, čipová karta, kryptografie, protokoly s nulovou znalostí, SmartDeck, revokace

ABSTRACT

The theoretical part of the bachelor thesis deals with the issue of secure attribute authentication, which places great emphasis on the protection of privacy and digital identity. The next part of the work deals with chip cards with MultOS operation system. The practical goal of this work is to create authentication application in C language using the attribute scheme CDDH19 and revocation scheme CDH16. The attribute scheme is based on signature scheme weak Boneh Boyen and on algebraic MAC funcion. For development it was used an interface Eclipse IDE with development environment SmartDeck. The final application will allow the users secure attribute authentication without any risks of leak of sensitive information. This application is implemented on MultOS chip card and tested in cooperation with the student, who programmed the verifier side on the microcomputer Raspberry PI.

KEY WORDS

attribut, authentication, MultOS, chip card, cryptography, zero-knowledge protocols, SmartDeck, revocation

BRODA, Jan. *Aplikace pro programovatelné čipové karty*. Brno, 2020. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/125911. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Jan Hajný.

PROHLÁŠENÍ

Prohlašuji, že svou bakalářskou práci na téma „Aplikace pro programovatelné 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 jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: ……. ……

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu doc. Ing. Janu Hajnému, Ph.D. za odborné vedení, trpělivost a podnětné návrhy při zpracování bakalářské práce. Také bych chtěl poděkovat panu Ing. Petru Dzurendovi, Ph.D. za cenné rady při tvorbě praktické části práce.

V Brně dne: ……. ……

OBSAH

ÚVOD

V současnosti, kdy vývoj technologií jde neustále dopředu, řada poskytovatelů nabízí své služby elektronicky. Pro přístup k těmto službám je mnohdy vyžadováno, aby uživatel prokázal svou identitu. Tímto roste riziko spojené se zneužitím či odcizením citlivých dat. Proto v dnešní době u takto distribuovaných služeb neustále rostou požadavky na ochranu soukromí a digitální identity. V zájmu poskytovatelů služeb je využití autentizačního systému, který omezí hrozbu spojenou se zneužitím citlivých dat. Pro tyto účely lze využít atributové autentizační systémy, jejichž problematikou se zabývá tato bakalářská práce.

V první kapitole práce je rozebrána problematika atributové autentizace a její využití. Pojednává se zde o základních kryptografických metodách a primitivech využívaných v atributové autentizaci. Dále jsou popsány protokoly, na kterých stojí námi implementované atributové schéma s revokací. Jedná se o podpisové schéma weak Boneh Boyen a algebraický MAC (Message Authentication Code) kód. Na závěr kapitoly je popsán samotný systém atributové autentizace s revokací, který je implementován v praktické části.

Druhá kapitola se zabývá úvodem do čipových karet a principem komunikace. Jsou zde popsány využívané transportní protokoly, struktury APDU (Application Protocol Data Unit) zpráv, ISO Cases a odpověď na reset.

Třetí kapitola pojednává o operačním systému MultOS. MultOS vyniká vysokou bezpečností, proto jí je věnována podstatná část této kapitoly. Kromě části o bezpečnosti a kryptografické podpoře je zde popsáno i rozdělení paměťového prostoru.

Čtvrtá kapitola popisuje nejčastěji využívané nástroje spojené s vývojem a správou aplikací na platformě MultOS. Jedná se především o nástroje SmartDeck a MUtil.

Pátá kapitola je věnována praktické části práce. Jsou zde popsány důležité postupy a výpočty spojené s vývojem aplikace pro atributovou autentizaci s revokací. Na konci kapitoly je vysvětleno nahrání aplikace na čipovou kartu MultOS a samotné otestování funkčnosti. Funkčnost byla ověřena ve spolupráci se studentem, který vytvořil stranu terminálu na jednodeskovém počítači Raspberry PI.

1 ATRIBUTOVÁ AUTENTIZACE

V současné době, kdy se běžně využívají elektronické služby a přístupové systémy, které pracují s citlivými osobními údaji, je v zájmu poskytovatelů služeb využívat autentizační metody, které jsou v souladu s neustále se navyšujícími požadavky na ochranu osobních dat. Především by měli respektovat nařízení o ochraně osobních údajů, jakými jsou například GDPR (General Data Protection Regulation) či eIDAS (electronic Identification, Authentication and trust Services) [1].

Cílem těchto autentizačních metod je omezit riziko spojené s odcizením či zneužitím digitální identity a citlivých dat. Proto by měli poskytovatelé služeb pracovat jen s nejdůležitějšími daty, které jsou potřebné pro správnou funkčnost autentizačního systému. Pro tyto účely lze využít atributovou autentizaci.

Atributová autentizace je založena na vlastnictví určitého atributu, kterým může být například řidičský průkaz. Tento atribut je uložen na specifickém nosiči, například čipové kartě, kterým uživatel prokazuje vlastnictví tohoto atributu. Pokud uživatel úspěšně prokáže držení daného atributu, ověřovatel zpřístupní uživateli požadovanou službu a ten ji může v systému anonymně využívat. Jestliže uživatel poruší pravidla stanovená systémem, může pomocí revokačního mechanismu dojít k vyřazení uživatele ze systému nebo eventuálně i k odhalení jeho identity.

Příklady využití atributové autentizace:

- Prokázání členství v určitém klubu – uživatel má přístup na golfové hřiště, jelikož prokáže, že platí členství v golfovém klubu.
- Držení řidičského průkaz – uživatel může řídit nákladní vozy, jelikož prokáže, že je držitelem platného řidičského oprávnění skupiny C.
- Přístup do laboratoří univerzity – uživatel může vstoupit do prostorů laboratoří, jelikož je schopen prokázat, že je studentem univerzity.
- Sleva na určité zboží – uživatel může nakupovat zboží se slevou, jelikož prokáže, že je vlastníkem slevové karty.
- Určení věku – uživatel si může koupit alkohol, jelikož prokáže, že je starší 18 let.

Entity atributového systému:

- Veřejná autorita – tato entita spravuje uživatele a v některých systémech může být rozdělena do dvou částí:
  - vydavatel – důvěryhodná entita, která vydává podepsané atributy jednotlivým uživatelům,
  - revokační autorita – tato autorita může uživateli zrušit platnost atributů nebo případně odhalit jeho identitu.
- Uživatel – entita využívající vlastního zařízení, pomocí kterého prokáže držení atributu.
- Ověřovatel – entita, která ověřuje, zdali uživatel je vlastníkem podepsaného atributu nutného pro přístup k dané službě.

Vlastnosti atributového systému:

- Nespojitelnost – ověřovatel nesmí být schopen poznat, zda jednotlivé verifikační relace patří jednomu uživateli.
- Nesledovatelnost – vydavatel nemůže identifikovat uživatele a jeho konkrétní přístupy, jelikož podepsané atributy jsou randomizovány.
- Anonymita – identita uživatele je během ověřování vlastnictví atributu skryta.
- Nepřenositelnost – každý uživatel má svůj jedinečný soukromý klíč, který nemůže poskytnout jinému uživateli.
- Selektivní odhalení atributů – každý uživatel si může vybrat, které atributy během procesu ověření odhalí. Zbylé atributy zůstávají skryty.
- Revokace – uživatel může být revokační autoritou vyřazen ze systému. K vyřazení uživatele ze systému může dojít například při ztrátě či odcizení čipové karty nebo pokud uživatel poruší pravidla stanovená poskytovatelem služeb. Revokační autorita má rovněž možnost odhalit uživatelovu identitu.

1.1 Kryptografie v atributové autentizaci

V této podkapitole budou popsány základní primitiva, která jsou využita v atributové autentizaci, a protokoly, na kterých je založena autentizační aplikace vytvořená v praktické části.

1.1.1 Kryptografické závazky

Kryptografické závazky [1, 2, 3] se využívají v autentizačních schématech, které kladou vysoký důraz na ochranu soukromých informací. Tyto závazky fungují na principu závazku k tajné informaci bez nutnosti zaslání této informace ověřovateli. Příkladem může být uživatel, který si vypočítá kryptografický závazek $c = commit(r, w)$, kde *commit* je závazkové schéma, r náhodné číslo a w tajná informace. Takto spočtený závazek již může zaslat ověřovateli, který není schopen zjistit původní podobu tajné informace w.

Vlastnosti kryptografických závazků:

- Skrytí – ověřovatel není schopen ze závazku zjistit tajnou informaci.
- Svázání – uživatel není schopen změnit tajnou informaci w a vypočítat stejný závazek c, tj. $c = commit(r, w) = commit(r', w')$: $w \neq w'$.

Mezi závazková schémata patří například závazkové schéma využívající diskrétního logaritmu, Pedersenovo závazkové schéma nebo závazkové schéma ElGamal. Všechna tato schémata jsou založena na problému diskrétního logaritmu.

1.1.2 Problém diskrétního logaritmu

Problém diskrétního logaritmu vychází z neschopnosti vypočítat inverzním postupem výraz, který byl spočten pomocí modulárního mocnění s vysokým prvočíslem, viz následující příklad:

Byla vygenerována multiplikativní grupa $\mathbb{Z}_p$, kde p je velké prvočíslo, q řád grupy, g generátor a hodnota $h \in \mathbb{Z}_q$. Z těchto hodnot je jednoduché modulárním mocněním spočítat výraz

$v = g^h \bmod p$. Avšak pokud by se z tohoto výrazu chtěla získat hodnota h, musel by se vyřešit diskrétní logaritmus $h = \log_g v \bmod p$, což je s využitím velkých prvočísel v praxi nerealizovatelné. Problém diskrétního logaritmu v současnosti využívají algoritmy, jakými jsou například Diffie-Hellman a DSA (Digital Signature Algorithm).

1.1.3 Protokoly s nulovou znalostí

Protokoly s nulovou znalostí [1, 2, 3] mají za úkol zamezit ztrátě důležitých soukromých informací (například klíčů nebo atributů) během autentizace. V těchto systémech uživatel poskytne důkaz o znalosti tajné informace, aniž by ji během procesu ověření prozradil. Tímto tato tajná informace zůstává skryta i před případným útočníkem. Jednoduchým příkladem využití protokolu s nulovou znalostí je autentizace pomocí nešifrovaného uživatelského jména a hesla. Ověřovatel nepotřebuje znát heslo, ale pouze důkaz toho, že ho zná uživatel. Proto uživateli stačí zaslat jen tento důkaz bez nutnosti odhalení dalších informací. Tímto se omezí riziko spojené se zneužitím soukromých informací.

Vlastnosti protokolů s nulovou znalostí:

- Spolehlivost – uživatel, který nezná tajnou informaci, není schopen přesvědčit ověřovatele, že tuto informaci zná.
- Úplnost – uživatel, který zná tajnou informaci, je vždy úspěšně ověřen.
- Nulová znalost – uživatel během ověření poskytuje ověřovateli pouze důkaz o znalosti tajné informace.

1.1.4 Sigma protokoly

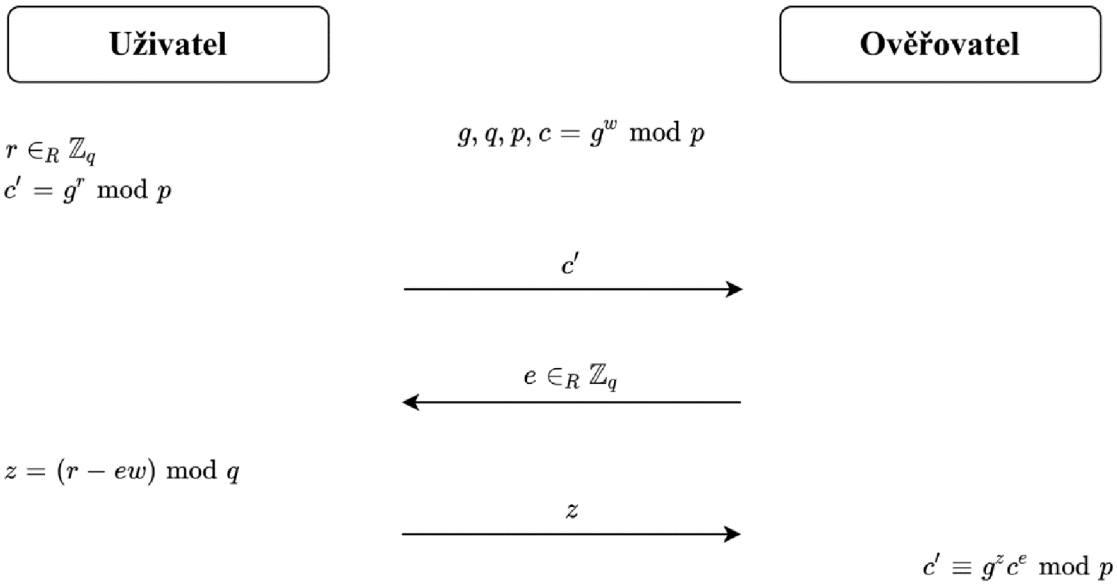
Pomocí Sigma protokolů [1] lze podobně jako u protokolů s nulovou znalostí prokázat držení tajné informace. Tyto protokoly jsou však oproti protokolům s nulovou znalostí rychlejší a účinnější, a proto mají v praxi větší uplatnění.

Vlastnosti Sigma protokolů:

- Třícestnost – protokol je založen na třech zprávách, které se posílají mezi uživatelem a ověřovatelem. První zprávou je závazek k tajné informaci a posílá se od uživatele k ověřovateli. Druhou zprávou je výzva a posílá se od ověřovatele k uživateli. Poslední zprávou je odpověď, která se zasílá od uživatele k ověřovateli. Ověřovatel je schopen na základě zpráv od uživatele ověřit znalost tajné informace.
- Spolehlivost – uživatel, který nezná tajnou informaci, není schopen správně odpovědět na výzvu ověřovatele a přesvědčit ho o opaku.
- Úplnost – uživatel, který zná tajnou informaci, je vždy schopen správně odpovědět na výzvu ověřovatele.
- Nulová znalost – během procesu ověření nedochází ke zveřejnění citlivých informací. Uživatel ověřovateli poskytuje pouze důkaz o znalosti tajné informace.

1.1.5 Důkaz znalosti diskrétního logaritmu

Využitím důkazu znalosti diskrétního logaritmu je uživatel schopen prokázat znalost tajné informace uvnitř kryptografického závazku bez nutnosti zveřejnění této informace. Uživatel s využitím modulární aritmetiky spočte veřejný závazek $c = g^w \bmod p$, kde g je generátor a p prvočíselný modulus. Na základě tohoto závazku je uživatel schopen prokázat, že zná diskrétní logaritmus $w = \log_g c \bmod p$, aniž by zveřejnil tajnou informaci w. Příkladem může být Schnorrův důkaz znalosti diskrétního logaritmu [1, 4] zobrazený na obr. 1.1.



Obr. 1.1: Schnorrův důkaz znalosti diskrétního logaritmu

Jsou známy veřejné parametry g, p a řád grupy q. Pomocí těchto parametrů se vypočte veřejný závazek $c = g^w \bmod p$. Uživatel vygeneruje náhodnou hodnotu r a spočte nový závazek $c' = g^r \bmod p$, který zašle ověřovateli. Ověřovatel pošle uživateli náhodnou hodnotu e, pomocí které uživatel vypočte hodnotu $z = (r - ew) \bmod q$. Tuto hodnotu zašle ověřovateli, který pomocí vztahu $c' \equiv g^z c^e \bmod p$ ověří, že uživatel zná tajnou informaci w, aniž by ji dokázal z přijatých hodnot získat.

1.1.6 Bilineární párování

Bilineární párování [1, 5] je mapování prvků, které je definováno jako $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$, kde $\mathbb{G}_1$, $\mathbb{G}_2$ a $\mathbb{G}_T$ jsou prvočíselné grupy řádu q. Pokud platí, že $\mathbb{G}_1 = \mathbb{G}_2$, jedná se o symetrické párování, a naopak pokud $\mathbb{G}_1 \neq \mathbb{G}_2$, jedná se o asymetrické párování.

Bilineární párování musí splňovat tyto požadavky:

- bilinearita – musí platit rovnice $e(g_1^x, g_2^y) = e(g_1, g_2)^{xy}$, kde $x, y \in \mathbb{Z}_q$, $g_1 \in \mathbb{G}_1$ a $g_2 \in \mathbb{G}_2$,
- nedegenerativnost – musí existovat generátory $g_1 \in \mathbb{G}_1$ a $g_2 \in \mathbb{G}_2$, pro které platí $e(g_1, g_2) \neq 1 \in \mathbb{G}_T$,

- spočitatelnost – musí existovat algoritmus, který je schopen vygenerovat parametry pro eliptickou křivku (q, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, e, g_1, g_2).

Z důvodu vysoké výpočetní náročnosti je doporučeno operaci bilineárního párování využívat na zařízeních s dostatečným výpočetním výkonem.

1.1.7 Podpisové schéma weak Boneh-Boyen

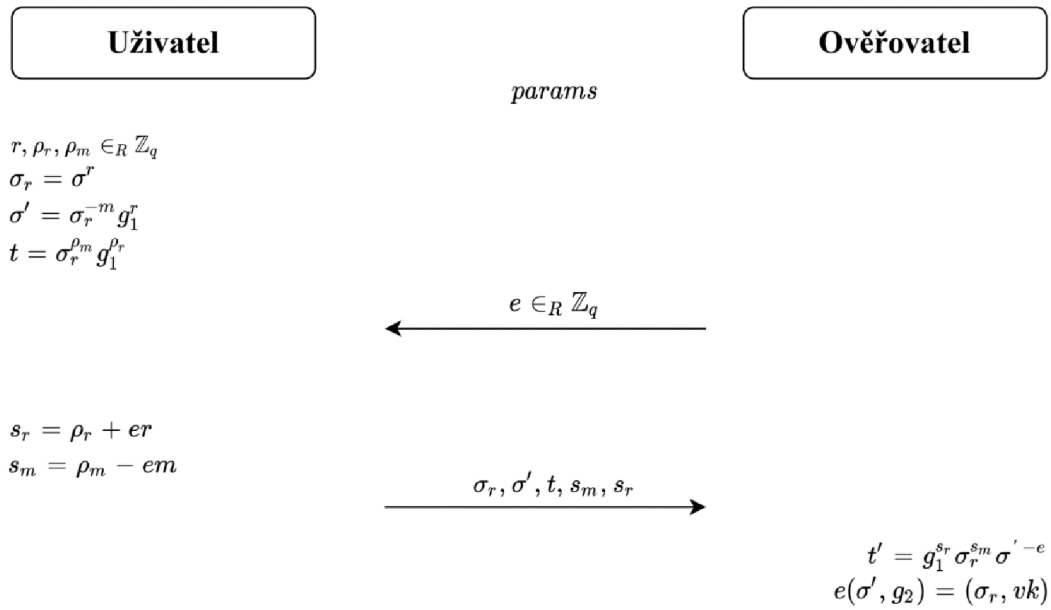
Schéma weak Boneh-Boyen (wBB) [1, 5] slouží k podpisu atributů. Výsledný podpis je možné uživatelem přepočítat tak, aby zde nebyla možnost ho spojit s původním podpisem a zároveň se zachovala jeho platnost. Díky tomu je docíleno:

- nespojitelnosti – ověřovatel nemůže zjistit, zda nově přepočítané podpisy patří jednomu uživateli,
- nesledovatelnosti – nově přepočítaný podpis nemůže být vydavatelem spojen s původním.

Schéma wBB je složeno z následujících algoritmů:

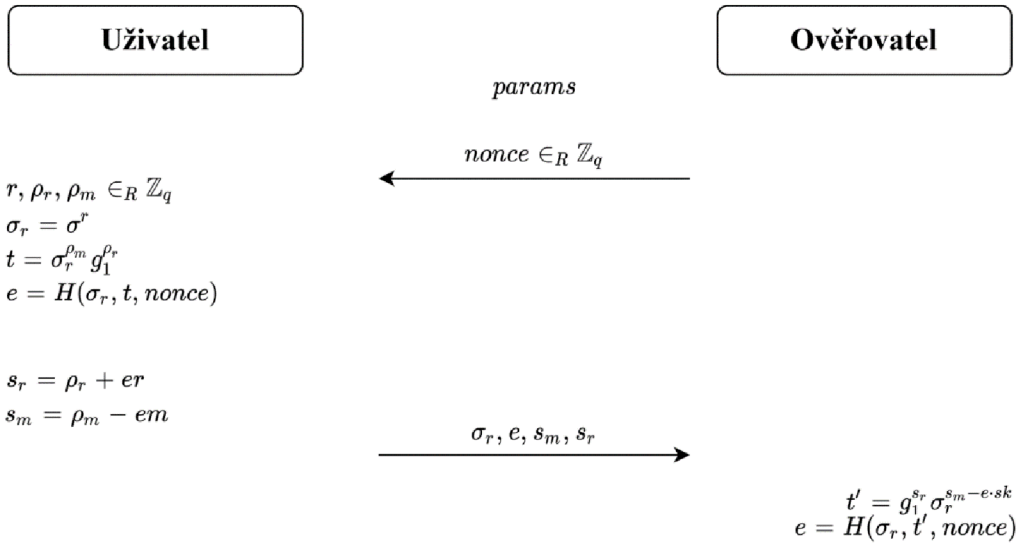
- `Setup`: Algoritmem `Setup` jsou vygenerovány parametry eliptické křivky s podporou bilineárního párování (q, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, e, g_1, g_2). Následně se vygeneruje náhodný soukromý klíč $sk \in_R \mathbb{Z}_q$ a vypočte veřejný klíč $vk = g_2^{sk}$. Na výstup jsou předány klíče sk a vk a systémové parametry $params(q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, g_1, g_2)$.
- `Sign`: Ze vstupního atributu $m \in \mathbb{Z}_q$ a soukromého klíče sk se spočte podpis $\sigma = g_1^{\frac{1}{sk+m}}$.
- `Verify`: Na základě podpisu σ, veřejného klíče vk a atributu m algoritmus ověří, zda platí $e(\sigma, vk) \cdot e(\sigma^m, g_2) = e(g_1, g_2)$. Pokud ano, je na výstup zaslána informace o platnosti podpisu.

Avšak takto spočtený podpis lze při procesu ověření spojit s uživatelem. To může být zneužito stranou ověřovatele, například pro mapování chování uživatele. Proto je využit randomizovaný wBB podpis, pomocí kterého uživatel dokáže prokázat znalost původního podpisu σ bez nutnosti jeho zveřejnění. Uživatel na základně náhodného čísla $r \in_R \mathbb{Z}_q$ vypočte nový podpis $\sigma_r = \sigma^r$ a hodnotu $\sigma' = \sigma_r^{-m} g_1^{r}$. Následně se pomocí Sigma protokolu vytvoří důkaz o znalosti původního podpisu. Důkaz lze ověřit pomocí párování $e(\sigma', g_2) = e(\sigma_r, vk)$. Důkaz o znalosti randomizovaného wBB podpisu je zobrazen na obr. 1.2.



Obr. 1.2: Důkaz znalosti weak Boneh-Boyen podpisu

Důkaz o znalosti wBB podpisu lze rovněž upravit na verzi, kdy není k ověření podpisu využito bilineárního párování. Důkaz o znalosti wBB podpisu ve verzi bez bilineárního párování je znázorněn na obr. 1.3.



Obr. 1.3: Důkaz znalosti wBB podpisu ve verzi bez párování

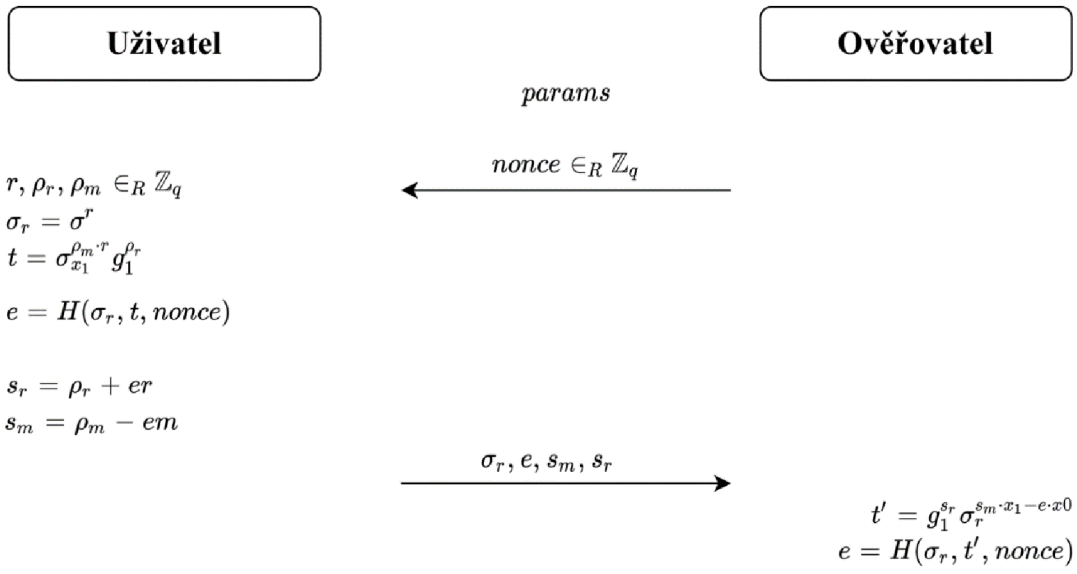
1.1.8 Algebraický MAC využívající weak Boneh-Boyen podpis

MAC je symetrický algoritmus [1, 6] zajišťující autentičnost a integritu zpráv. K sestavení MAC kódu se obvykle využívají blokové šifry nebo hashovací funkce. Takto sestavené MAC kódy neobsahují algebraickou strukturu, a proto je nelze účinně kombinovat s důkazy nulových znalostí. Z toho důvodu je potřeba využít algebraický MAC, jehož konstrukce je založena na algebraických operacích.

Kombinace algebraického kódu MAC a podpisového schématu wBB je založena na následujících algoritmech:

- `Setup`: Algoritmem `Setup` jsou vygenerovány parametry eliptické křivky s podporou bilineárního párování (q, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, e, g_1, g_2). Pro n atributů jsou vygenerovány náhodné hodnoty (x_0, …, x_n) $\in_R \mathbb{Z}_q$ jako klíč sk a spočteny (X_0, …, X_n) = ($g_2^{x_0}$, …, $g_2^{x_n}$). Na výstup jsou předány systémové parametry $params$ (q, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, e, g_1, g_2, (X_0, …, X_n)) a klíč sk = (x_0, …, x_n).
- `Sign`: Ze vstupních atributů (m_1, …, m_n) $\in \mathbb{Z}_q$ a klíče sk se spočte podpis $\sigma = g_1^{\frac{1}{x_0+x_1m_1+\ldots+x_nm_n}}$ a pomocné hodnoty (σ_{x_0}, …, σ_{x_n}) = (σ^{x_0}, …, σ^{x_n}).
- `Verify`: Ze vstupního podpisu σ, klíče sk a atributů (m_1, …, m_n) algoritmus ověří, zda platí $g_1 = \sigma^{x_0+m_1x_1+\ldots+x_nm_n}$. Pokud ano, je na výstup zaslána informace o platnosti podpisu.

Důkaz o znalosti randomizovaného kódu MAC lze prokázat obdobně jako u podpisu wBB. Vygeneruje se náhodné číslo $r \in_R \mathbb{Z}_q$, spočte nový randomizovaný podpis $\sigma_r = \sigma^r$ a hodnoty ($\sigma_{x_0}^r$, …, $\sigma_{x_n}^r$). Pomocí Sigma protokolu se vytvoří důkaz o znalosti podpisu a bilineárním párováním ($\sigma_{x_i}^r$, g_2) = (σ_r, X_i) se ověří. Rovněž jako u wBB podpisu lze využít verzi bez bilineárního párování. Důkaz znalosti randomizovaného kódu MAC s wBB podpisem pro jeden atribut ve verzi bez bilineárního párování je ilustrován na obr. 1.4.



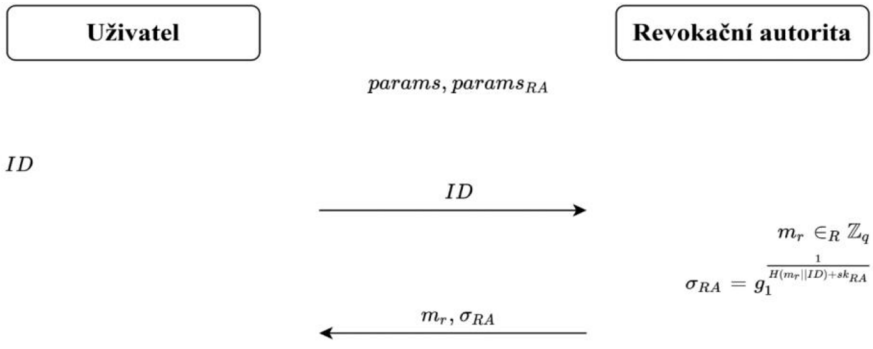
Obr. 1.4: Důkaz znalosti kódu MAC s wBB podpisem ve verzi bez bilineárního párování

1.2 Systém atributové autentizace s revokací

Systém atributové autentizace [1], který je implementován v rámci praktické části, je založen na kryptografických algoritmech a primitivech popsaných v předcházejících částech. Systém zahrnuje možnost revokace uživatele a splňuje všechny bezpečnostní vlastnosti pro ochranu soukromí, tj. anonymitu, nespojitelnost, nesledovatelnost, nepřenositelnost, selektivní odhalení atributů a revokaci. Tyto vlastnosti jsou blíže popsány v kapitole 1.

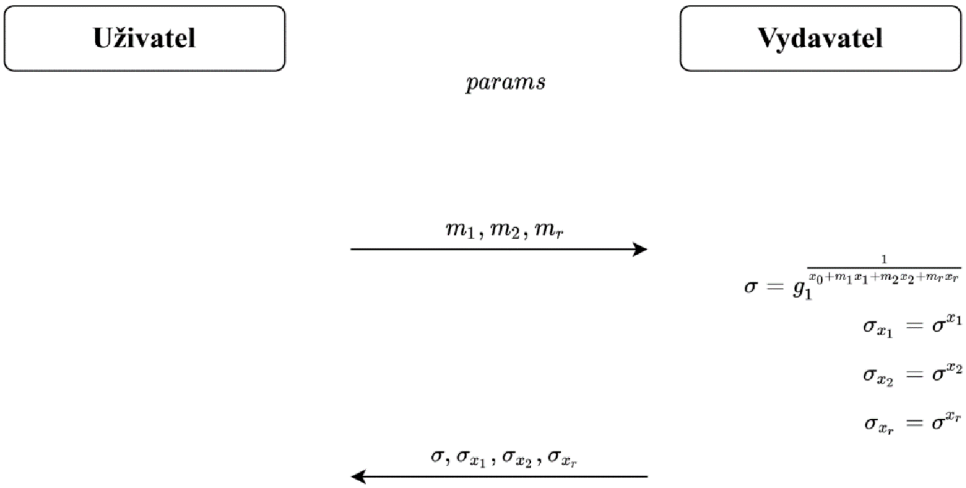
Jednotlivé entity schématu (vydavatel, ověřovatel, uživatel a revokační autorita) spolu kooperují pomocí následujících částí:

- `Setup`: Pomocí algoritmu `Setup` vydavatel vygeneruje všechny veřejné parametry, které budou následně zahrnuty na vstupech ostatních algoritmů. Konkrétním výstupem algoritmu `Setup` jsou parametry eliptické křivky *params* (q, $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$, e, g_1, g_2) a klíč vydavatele $sk = (x_0, \ldots, x_n) \in_R \mathbb{Z}_q$.
- `SetupRA`: Tento algoritmus na základě maximálního počtu verifikací v daném časovém úseku a parametrů grupy *params* vygeneruje všechny veřejné parametry a klíče revokační autority. Maximální počet verifikací v_{max} je spočten jako $v_{max} = k^j$, kde k a j jsou celá čísla. Pro potřeby bakalářské práce je maximální počet verifikací za jednu časovou epochu omezen na sto, tj. $v_{max} = 10^2$. Následně se vygenerují náhodná čísla $(\alpha_1, ..., \alpha_j) \in_R \mathbb{Z}_q$ (tj. $(\alpha_1, \alpha_2) \in_R \mathbb{Z}_q$) a spočtou hodnoty $h_z = g_1^{\alpha_z}$, kde $z \in (1, \ldots, j)$, tj. $z \in (1, 2)$. Klíče revokační autority jsou vygenerovány jako $sk_{RA} \in_R \mathbb{Z}_q$ a $vk_{RA} = g_2^{sk_{RA}}$, kde sk_{RA} je soukromým klíčem a vk_{RA} veřejným klíčem. Dále jsou vygenerovány náhodné elementy $(e_1, \ldots, e_k) \in_R \mathbb{Z}_q$ (tj. $(e_1, \ldots, e_{10}) \in_R \mathbb{Z}_q$) a vypočteny podpisy $\sigma_{e_1-e_k} = \frac{1}{e_{1-k}+sk_{RA}}$ pro každý z těchto elementů, tj. $\{(e_1, \sigma_{e_1}), ..., (e_{10}, \sigma_{e_{10}})\}$. Na závěr jsou vytvořeny dva prázdné revokační seznamy, revokační seznam RL a seznam revokačních atributů RH.
- `Issue`: Algoritmus `Issue` slouží k vydání atributů uživateli a je rozdělen do dvou částí. V první části běží mezi uživatelem a revokační autoritou, viz obr 1.5. Uživatel zašle revokační autoritě svoje ID a ta vygeneruje revokační atribut $m_r \in_R \mathbb{Z}_q$ a vytvoří na něj podpis σ_{RA}. Následně aktualizuje svůj seznam revokačních atributů $RH = RH + m_r \| \text{ID}$ a zašle hodnoty m_r a σ_{RA} uživateli, který je uloží.



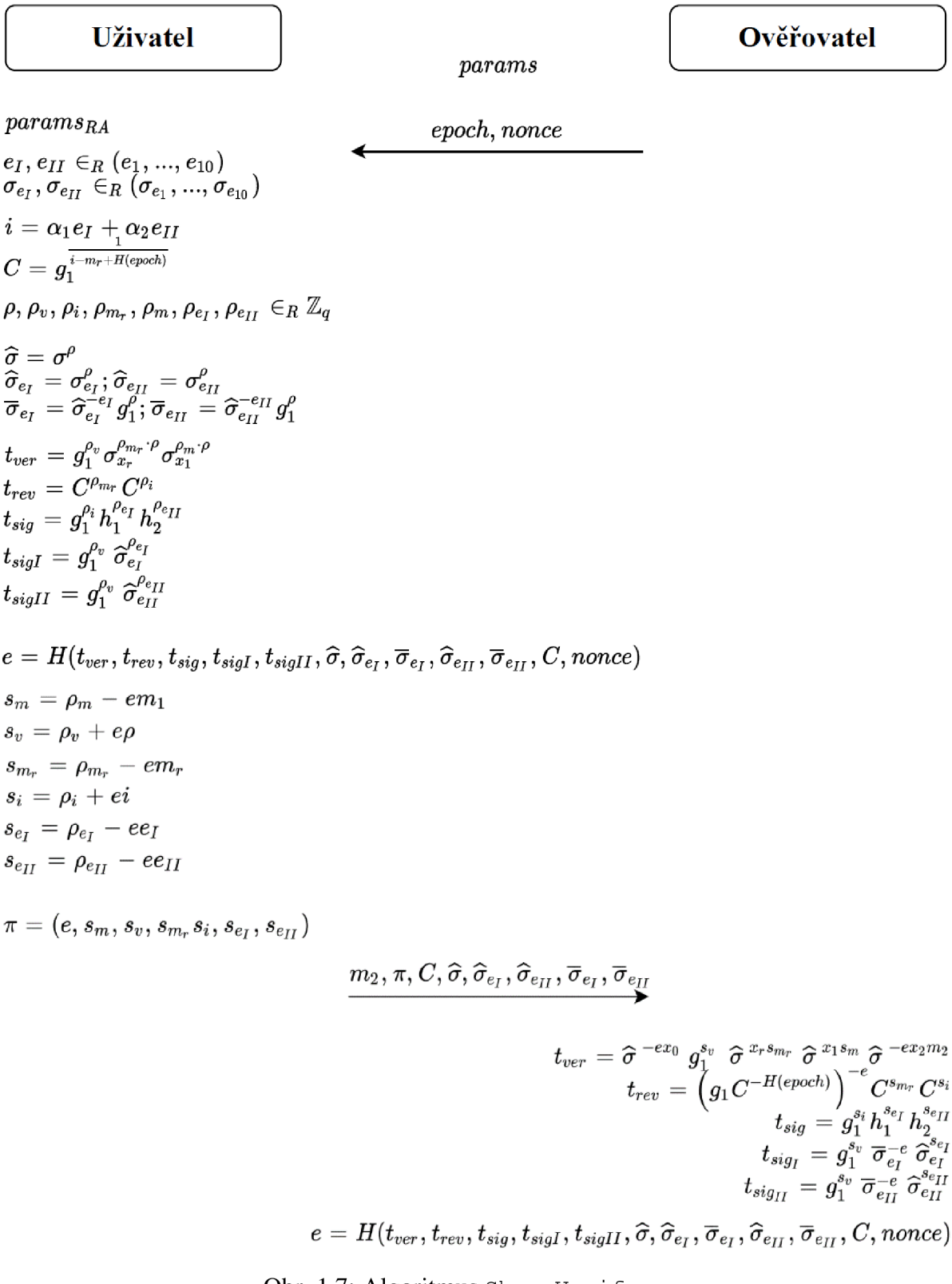
Obr. 1.5: Algoritmus `Issue` – vydání m_r, σ_{RA}

V druhé části algoritmus běží mezi uživatelem a vydavatelem, viz obr. 1.6. Uživatel zašle vydavateli všechny své atributy společně s revokačním atributem m_r a podpisem σ_{RA}. Vydavatel na přijaté atributy vytvoří příslušné podpisy ($\sigma, \sigma_{x_0}, \ldots, \sigma_{x_{n-1}}, \sigma_{x_r}$) a ty následně zašle uživateli, který je uloží. V praktické části jsou uživatelem využity dva atributy, m_1 jako skrytý atribut a m_2 jako odhalený atribut. Vydání podpisů pro dané atributy je znázorněno na obr. 1.6.



Obr. 1.6: Algoritmus `Issue` – vydání MAC kódů na atributy

- `Show-Verify`: Slouží pro prokázání vlastnictví atributů a běží mezi uživatelem a ověřovatelem. Uživatel na základě jedinečných kombinací vygenerovaných randomizérů ($e_1, \ldots, e_{10}$) spočte hodnotu $i=\alpha_1 e_{1-10}+\alpha_2 e_{1-10}$, kterou společně s hashem aktuální časové epochy $H(epoch)$ a revokačního atributu m_r použije k výpočtu revokačního pseudonymu $C = g_1^{\frac{1}{i-m_r+H(epoch)}}$. Uživatel následně randomizuje svůj podpis na atributy a vytvoří důkaz znalosti π, který je rozšířen o důkaz, že při výpočtu revokačního pseudonymu C byly použity platné hodnoty. Na závěr uživatel zašle sestavený důkaz ověřovateli, který ho ověří a zároveň zkontroluje, zda se na seznamu RL nenachází revokační pseudonym C. Konkrétní podoba algoritmu `Show-Verify`, která byla implementována v praktické části, je znázorněna na obr. 1.7.



Obr. 1.7: Algoritmus `Show-Verify`

- `Revoke`: Posledním algoritmem je algoritmus `Revoke` běžící mezi ověřovatelem a revokační autoritou, jehož pomocí lze ze systému vyřadit uživatele, popř. odhalit jeho identitu. Revokační autorita spočte všechny pseudonymy *C*, které je schopen vypočítat uživatel, a následně tyto pseudonymy zahrne do revokačního sezamu $RL=RL+C$, který předá na výstup.

2 ÚVOD DO ČIPOVÝCH KARET

Již několik desítek let jsou karty využívány jako oblíbený platební prostředek a jednoduché úložiště informací. Před nasazením integrovaných čipů karty disponovaly bezpečnostními prvky, jakými byly například informace o majiteli karty vyražené do plastu, informace zakódované v magnetickém proužku nebo speciální tisk viditelný pouze pod ultrafialovým zářením [7]. Problémem těchto základních bezpečnostních prvků je jejich náchylnost ke zneužití. Útočník, který by kartu odcizil, nebo vytvořil klon, by jednoduše získal přístup k všem službám.

Čipové karty se zabudovaným integrovaným čipem, někdy také označované jako ICC (Integrated Circuit Card) či smart cards, se díky své bezpečnosti hojně využívají nejen pro rychlé a bezpečné zpracování bankovních transakcí, ale i k ověření identity v elektronických přístupových systémech či přístupu k elektronickým službám. Slouží také jako bezpečné úložiště soukromých informací a klíčů. Integrovaný čip obsahuje tři druhy pamětí:

- RAM (Random Access Memory) – napěťově závislá dynamická paměť sloužící k dočasnému ukládání dat,
- ROM (Read-Only Memory) – napěťově nezávislá paměť, ve které je uložen operační systém,
- EEPROM (Electrically Erasable Programmable Read-Only Memory) – napěťově nezávislá paměť sloužící k ukládání aplikací.

Většina čipových karet disponuje jen jednou funkcí, která je dána určitou společností, například bankou. V tomto případě může být čipová karta využita jen k účelům stanoveným bankou. Aplikaci, která je nahrána na čipové kartě s podporou jedné aplikace, je velmi obtížné změnit. Pokud je potřeba, aby čipová karta obsahovala více funkcí nezávislých na sobě, například podporu bankovních transakcí a ověření přístupu v přístupovém systému, je zde možnost využít multiaplikační čipové karty, které umožnují nahrát více aplikací na jeden integrovaný čip. Jelikož se jednotlivé společnosti musí dělit o prostředky integrovaného čipu, je zde kladen vysoký důraz na bezpečnost.

2.1 Princip komunikace

Komunikace je založena na modelu klient – server, kde terminál reprezentuje stranu klienta a čipová karta stranu serveru. Aplikace terminálu a čipové karty mezi sebou komunikují na základně výměny APDU zpráv. Terminál na kartu zasílá APDU příkazy a ta na ně odpovídá APDU odpověďmi. Maximální velikost APDU zpráv je stanovena zvoleným transportním protokolem TPDU (Transmission Protocol Data Unit).

Nejčastěji využívané transportní protokoly jsou [8]:

- T = 0 – poloduplexní, bajtově orientován, přenášející pole bajtů s maximální velikostí řídících dat 255 bajtů,

- T = 1 – poloduplexní, blokově orientován, přenášející bloky bajtů s maximální velikostí řídících dat 65 535 bajtů.

2.1.1 Struktura APDU příkazu

Struktura APDU příkazu se skládá ze dvou částí, viz tab. 2.1. Povinnou částí je záhlaví o velkosti čtyř bajtů obsahující komponenty, které jsou označovány jako:

- CLA (Class) – udává třídu příkazů a odpovědí,
- INS (Instruction) – udává instrukci, která má být zpracována,
- P1, P2 (Parameters 1, 2) – slouží k upřesnění instrukce.

Volitelná část má proměnnou délku a obsahuje řídící DATA, jejichž maximální velikost se odvíjí od použitého transportního protokolu. Dále obsahuje komponenty:

- Lc (Length command) – udává délku dat v komponentu DATA, která budou zaslána čipové kartě,
- Le (Length expected) – očekávaná délka dat, která mají být vrácena po zpracování příkazu čipovou kartou.

Tab. 2.1: Struktura APDU příkazu

Povinné záhlaví				**Volitelná datová část**		
CLA	INS	P1	P2	Lc	DATA	Le

2.1.2 Struktura APDU odpovědi

APDU odpověď je obdobně jako APDU příkaz složena ze dvou částí, viz tab. 2.2. Povinnou částí je zápatí, které obsahuje komponenty SW1 a SW2. Tyto komponenty slouží jako návratové kódy upřesňující stav zpracovaného příkazu, viz tab. 2.3. Volitelná část obsahuje data vrácená kartou.

Tab. 2.2: Struktura APDU odpovědi

Volitelná datová část	**Povinné zápatí**	
DATA	SW1	SW2

Tab. 2.3: Příklady návratových kódů

Návratový kód		**Popis**
SW1	**SW2**	
61	XX	Zpracování proběhlo úspěšně, XX bajtů dat je připraveno k vyzvednutí
90	00	Příkaz úspěšně zpracován
6C	XX	Chybná hodnota délky Le, XX reprezentuje správnou hodnotu
6E	00	Nepodporovaná třída
6D	00	Chybný či nepodporovaný kód instrukce

Kompletní seznam návratových kódů je uveden v pramenu [9].

2.1.3 ISO Cases

Během vývoje aplikací na čipových kartách je nutno pracovat s tzv. ISO Cases, které definují struktury APDU příkazů a odpovědí, viz tab. 2.4.

Tab. 2.4: ISO Cases

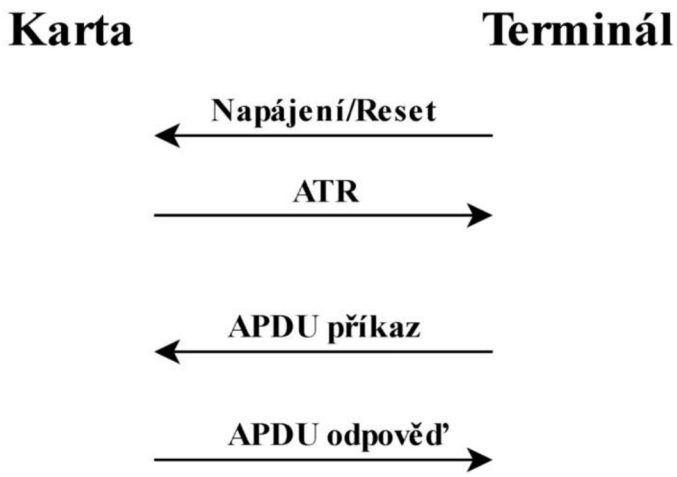
Case	Struktura APDU příkazu		Struktura APDU odpovědi	
	Povinné záhlaví	Volitelná datová část	Volitelná datová část	Povinné zápatí
1	CLA INS P1 P2	-	-	SW1 SW2
2	CLA INS P1 P2	Le	DATA	SW1 SW2
3	CLA INS P1 P2	Lc DATA	-	SW1 SW2
4	CLA INS P1 P2	Lc DATA Le	DATA	SW1 SW2

2.1.4 Odpověď na reset

Před samotnou výměnou APDU zpráv je (po připojení čipové karty k napájení, tj. vložením karty do terminálu) vyslán příkaz Reset, na který karta reaguje ATR (Answer To Reset) odpovědí [10], viz obr. 2.1.

MultOS rozděluje ATR odpovědi na primární a sekundární [7]. Prvotním připojením karty k napájení se vyšle primární ATR. Sekundární ATR je vysláno pokaždé, když čipová karta obdrží příkaz Reset. ATR odpověď může mít velikost až 33 bajtů, avšak obsah odpovědi zřídka zaplní všech 33 bajtů. ATR odpověď se skládá ze dvou částí:

- Historical characters – výrobcem definované pole, které může sloužit pro uchování informací o kartě, výrobci, operačním systému, ale například i o stavu elektronické peněženky.
- Interface characters – obsahují informace o parametrech komunikace včetně typu transportního protokolu (T=1, T=0), úrovně napětí a proudu EEPROM paměti a frekvenci hodinového signálu integrovaného čipu.



Obr. 2.1: ATR a výměna APDU zpráv

3 MULTOS

MultOS je široce rozšířený, multiaplikační operační systém, který byl primárně vytvořen pro čipové karty. Čipové karty s operačním systémem MultOS jsou hojně využívány různými společnostmi, a to především pro jejich vysokou bezpečnost.

3.1 Rozdělení paměti

Paměť pro aplikace je rozdělena na kódový a datový prostor [7]. Kódový prostor je vyhrazen v EEPROM paměti a je v něm uložen kód aplikace. Do tohoto prostoru nelze vstoupit za účelem čtení nebo zápisu. Kód je interpretován pomocí virtuálního stroje AAM (Application Abstract Machine). Datový prostor obsahuje data aplikace a je rozdělen do tří částí:

- Dynamická paměť (`melsession`) – slouží k dočasnému ukládání dat aplikace do RAM paměti. Data jsou v této paměti uložena po dobu napájení karty. Dynamická paměť rovněž obsahuje zásobník. Ten je například využit při programování aplikace s využitím MEL (MultOS Executable Language) assembler kódu. Před nahráním aplikace na kartu je nutno určit velikost dynamické paměti.
- Veřejná paměť (`melpublic`) – společná RAM paměť pro potřeby aplikace čipové karty a terminálu. Jsou zde uloženy příchozí APDU příkazy a odchozí APDU odpovědi. K adresaci paměťového prostoru jsou využity dva registry. Prvním je Public Top registr sloužící pro ukládání záhlaví APDU příkazu a zápatí APDU odpovědi. Druhým je Public Bottom registr, ve kterém jsou uložena data APDU příkazu a odpovědi. Data APDU odpovědi se terminálu zasílají od nultého indexu Public Bottom registru. U dat uložených ve veřejné paměti je po dobu běhu aplikace zajištěna jejich důvěrnost. Aby se aplikace vyvarovala úniku citlivých dat, je nutno tato data odstranit z veřejné paměti ještě před ukončením aplikace.
- Statická paměť (`melstatic`) – slouží k ukládání kódu a statických dat aplikace do EEPROM paměti. Přístup k těmto datům má přístup pouze příslušná aplikace. K adresaci jsou využity registry Static Top a Static Bottom. Paměťový prostor pro aplikaci začíná nultým indexem Public Bottom registru. Při vykonávání instrukce zápisu dat může nastat problém, že se data nezapíšou do paměti celá. To může způsobit narušení statické paměti. K vyřešení tohoto problému je využit Data Item Protection. Tento mechanismus zajistí, že se instrukce zápisu provede buď celá, nebo se neprovede vůbec.

3.2 Bezpečnost aplikací

Každá MultOS čipová karta může obsahovat aplikace různých společností, které fungují nezávisle na sobě. Je vyžadováno, aby nedocházelo k vzájemnému přístupu do prostor, kde jsou uložena citlivá data každé aplikace. Tento prostor je nutné pro každou nahranou aplikaci vymezit a zabezpečit. Virtuální stroj vyhradí každé aplikaci určitou pevnou velikost paměti, ve které je nezávisle uložen datový a kódový prostor aplikace. Tyto části paměti jsou od sebe bezpečně

izolovány firewallem. Pokud by jedna aplikace chtěla přistoupit do paměťového prostoru jiné aplikace, tak virtuální stroj tuto aplikaci ukončí.

3.3 Bezpečnost při nahrávání aplikací

Bezpečnost při nahrávání aplikace je založena na asymetrické kryptografii, tedy na principu veřejných a privátních klíčů. Po aktivaci MultOS čipové karty jsou tyto klíče vygenerovány pomocí KMA (Key Management Authority) autority a následně nahrány na kartu.

3.3.1 Bezpečnost ALU

Vytvořená aplikace musí být před nahráním na kartu zapouzdřena do ALU (Application Load Unit) jednotky [11], která se dle úrovně zabezpečení dělí na:

- Unprotected (nechráněná) – obsahuje pouze kód aplikace.
- Protected (chráněná) – obsahuje kód aplikace a digitální podpis pro zaručení autentičnosti.
- Confidential (šifrovaná) – obsahuje kód aplikace, digitální podpis a KTU (Key Transformation Unit) sloužící pro šifrování ALU. Šifruje se pomocí veřejného klíče cílové karty. Dešifrovat ALU lze pomocí soukromého klíče cílové karty. K šifrování je využita asymetrická šifra RSA (Rivest-Shamir-Adleman).

Využitím digitálního podpisu a šifrování lze ALU distribuovat přes jakoukoliv bezpečnou či nebezpečnou síť bez rizika úniku kódu aplikace.

3.3.2 Certifikace

Každá aplikace, která má být nahrána na kartu, musí obsahovat certifikát ALC (Application Load Certificate) [12]. Certifikát slouží k zajištění integrity a autentičnosti aplikace. Vydavateli čipové karty rovněž poskytují kontrolu nad nahrávanými aplikacemi. Vydavatel karty o něj musí zažádat u certifikační autority KMA, která tuto aplikaci zaregistruje. Po registraci aplikace touto certifikační autoritou je možno aplikaci nahrát na kartu. Při nahrávání integrovaný čip ověří přítomnost certifikátu pomocí veřejného klíče KMA. Odstranění nahrané aplikace se provádí pomocí ADC (Application Delete Certificate), o který rovněž může vydavatel zažádat u KMA po registraci aplikace.

3.4 Kryptografická podpora MultOS

Aplikace vytvářené pro platformu čipových karet MultOS jsou většinou založeny na kryptografických operacích a algoritmech. Platforma MultOS od verze 4.2 podporuje dostatek kryptografických operací a algoritmů potřebných pro vytvoření bezpečných kryptografických systémů. Mezi základní podporované kryptografické operace a algoritmy patří:

- modulární aritmetika – modulární násobení, modulární mocnění, modulární inverze, modulární redukce,
- hashovací funkce – SHA1(Secure Hash Algorithm) a SHA2 (SHA-224, SHA-256, SHA-384, SHA-512),

- asymetrické algoritmy – RSA, ECDH (Elliptic Curve Diffie Hellman), ECDSA (Elliptic Curve Digital Signature Algorithm) a ECIES (Elliptic Curve Integrated Encryption Scheme),
- symetrické algoritmy – AES (Advanced Encryption Standard), DES (Data Encryption Standard) a 3DES (Triple Data Encryption Standard).

Kompletní seznam podporovaných operací je uveden v pramenu [13].

3.4.1 Podpora eliptických křivek

Eliptické křivky jsou často využívané algebraické struktury pro tvorbu kryptografických systémů. Vysoké oblibě vděčí především pro jejich bezpečnost a rychlost při využití kratších klíčů. Podpora eliptických křivek na platformě MultOS je od verze 4.2 a zahrnuje operace jako sčítání bodů, násobení bodů, inverzi bodu, změnu reprezentace bodu, test rovnosti bodů, ověření platnosti bodu, generování podpisu a ověření podpisu. Eliptická křivka je na platformě MultOS definovaná nad prvočíselným polem $\mathbb{F}p$ s maximální velikostí 512 bitů. Souřadnice bodu křivky jsou reprezentovány buď v afinním (x, y), nebo projektivním (x, y, z) tvaru.

4 NÁSTROJE PRO VÝVOJ APLIKACÍ NA PLATFORMĚ MULTOS

Pro vývoj aplikací na MultOS čipové karty je možno využít především jazyk C, Javu nebo MEL assembler kód. Pokud je pro vývoj využit jazyk C nebo Java, je nutné přeložit takto napsaný kód do MEL. Ke zpracování takto přeložených instrukcí slouží virtuální stroj AAM.

4.1 MultOS SmartDeck

K vývoji aplikací na čipových kartách MultOS slouží vývojové prostředí SmartDeck [14] běžící na platformách Microsoft Windows. Distribuce vývojového prostředí SmartDeck je pod záštitou MultOS konsorcia a lze jej zadarmo získat na oficiálních stránkách MultOS [15]. SmartDeck zakomponovaný v rozhraní Eclipse IDE umožňuje vývojářům efektivně využít všech nástrojů pro vývoj, správu, překlad a ladění aplikací. Pro správu vytvořených aplikací slouží zabudovaná komponenta `hterm.exe`. Tato komponenta se především využívá k nahrávání a odstranění aplikací. Avšak lze ji použít i pro zasílání APDU zpráv, správu certifikátů, výpis informací o operačním systému nahraném na kartě nebo k výpisu informací o čipové kartě a nahrané aplikaci. Dalšími důležitými komponenty vývojového prostředí SmartDeck jsou:

- `halugen.exe` – slouží k vytvoření ALU ze spustitelných souborů s příponou `.hzx`,
- `mdb.exe` – slouží jako ladící rozhraní mezi Eclipse IDE a simulátorem `hsim.exe`,
- `hls.exe` – slouží k výpisu užitečných informací například paměťových sekcí souborů s příponou `.hzx`,
- `hsim.exe` – simulátor MultOS virtuálního stroje,
- `melcertgen.exe` – slouží ke generování certifikátů pro nahrávání a odstraňování aplikací u karet určených pro vývojáře.

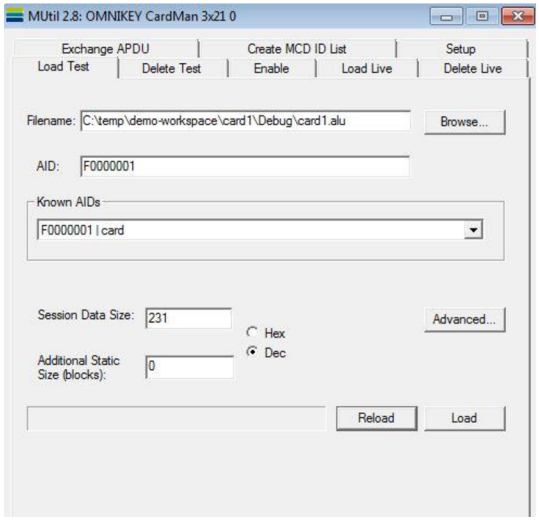
4.2 MUtil

MUtil je multifunkční nástroj [16] pro práci s vývojářskými nebo klasickými tzv. live MultOS čipovými kartami. Je volně dostupný ke stažení na stránkách MultOS [15]. Obdobně jako u SmartDeck komponentu `hterm.exe` lze s jeho pomocí nahrávat a odstraňovat aplikace. Při použití vývojářských karet není nutné nahrávat certifikáty ALC a ADC. Rovněž je zde podpora komunikace s aplikací karty na základě výměny APDU zpráv, díky které lze otestovat, zda čipová karta správně komunikuje se čtečkou. Pomocí MUtil lze také získat mnoho informací o čipové kartě a operačním systému. Je možné zobrazit například verzi MultOS, nahrané aplikace, informace o výrobci nebo ATR odpověď.

4.2.1 Nahrání aplikace

Nahrání aplikace na vývojářskou čipovou kartu se provádí v záložce *Load Test,* viz obr. 4.1. V této záložce je potřeba v poli *Filename* definovat cestu k ALU souboru, který má být na kartu nahrán.

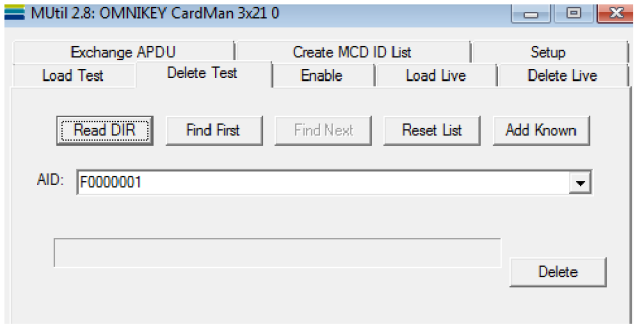
Tento soubor lze získat pomocí SmartDeck komponentu `halugen.exe`, který ze souboru s příponou `.hzx`[1] vytvoří soubor s příponou `.alu`. Dále je potřeba v okně *AID* (Application IDentifier) vyplnit identifikátor aplikace. Vyplňování pole *AID* lze usnadnit pomocí pole *Known AIDs*, do kterého je možné uložit seznam využívaných identifikátorů. Tento seznam lze editovat pomocí konfiguračního souboru `MUtil.ini`. Na závěr je potřeba specifikovat velikost RAM paměti v poli *Session Data Size*. Velikost RAM paměti lze zjistit pomocí SmartDeck komponentu `hls.exe`. Tuto hodnotu lze nástroji MUtil předat v decimální či hexadecimální podobě. Po specifikaci všech těchto potřebných částí je již možno aplikaci úspěšně nahrát na čipovou kartu.



Obr. 4.1: MUtil – záložka *Load Test*

4.2.2 Odstranění aplikace

Odstranění aplikace na vývojářské čipové kartě je možné v záložce *Delete Test*, viz obr. 4.1. Je zde potřeba specifikovat identifikátor aplikace, která má být odstraněna. Identifikátor nahrané aplikace lze zjistit pomocí možnosti *Read DIR*, kdy se přečte obsah adresáře DIR, ve kterém je obsažen identifikátor nahrané aplikace.



Obr. 4.2: MUtil – záložka *Delete Test*

[1] Spustitelný soubor s příponou .hzx se v Elipse IDE vytvoří pomocí tlačítka *Build Project*.

5 APLIKACE PRO ATRIBUTOVOU AUTENTIZACI

Praktická část této bakalářské práce je zaměřena na vývoj aplikace pro atributovou autentizaci s revokací na straně uživatele, tj. čipové karty s operačním systémem MultOS. Aplikace je vytvořena pro jeden zveřejněný a jeden nezveřejněný atribut. Všechny dílčí algoritmy implementovaného atributového schématu jsou popsány v kapitole 1.2. Jednotlivé proměnné, které bylo potřeba na straně karty vypočítat, jsou zobrazeny na obr. 1.7 a postup jejich výpočtů bude popsán v této kapitole. Pro vývoj byl využit virtuální stroj s operačním systémem Windows 7, který obsahuje přednastavené rozhraní Eclipse IDE a vývojové prostředí SmartDeck. Výsledná aplikace byla na čipovou kartu nahrána pomocí nástroje MUtil. Otestování funkčnosti proběhlo ve spolupráci se studentem, který vytvořil stranu ověřovatele na zařízení Raspberry PI.

5.1 Deklarace atributů, knihoven a instrukcí

Na začátku zdrojového kódu jsou definovány atributy, které slouží pro specifikaci určitých parametrů aplikace. Zejména byly deklarovány následující atributy:

```
#pragma attribute("aid", "f0 00 00 01")
```

Atribut sloužící ke specifikaci AID aplikace.

```
#pragma attribute("dir", "61 10 4f 4 f0 00 00 01 50 8 65 6c 6f 79 61
6c 74 79")
```

Tento atribut nastaví záznam v DIR. Obsahuje především AID aplikace a název.

Velice důležitými atributy jsou ty, které upřesní, do jaké části paměti se zapíšou využívané datové struktury. Popis těchto částí paměti je uveden v podkapitole 3.1:

```
#pragma melstatic //EEPROM - Static memory
#pragma melsession //RAM - Session memory
#pragma melpublic //RAM - Public memory
```

Knihovny potřebné pro vývoj autentizační aplikace jsou součástí vývojového prostředí SmartDeck. Jedná se především o tyto knihovny:

```
#include <multoscomms.h>
```

Knihovna zajišťující komunikaci mezi čipovou kartou a terminálem.

```
#include <multoscrypto.h>
```

Knihovna obsahující kryptografické operace podporované operačním systémem MultOS.

```
#include <string.h>
```

Knihovna pro práci s pamětí. V aplikaci je využita především pro funkci `memcpy` a `memcmp`.

```
#include <multosarith.h>
```

Knihovna pro podporu aritmetických operací.

Instrukce aplikace jsou pro větší přehlednost definovány pomocí symbolických konstant. Aplikace obsahuje zejména tyto instrukce:

```
#define INS_GET_ID          0x05
```

Instrukce, pomocí které uživatel zašle revokační autoritě své ID. Jako ID může být použito například číslo občanského průkazu. Tato instrukce je součástí algoritmu `Issue`.

```
#define INS_SET_RA_PARAMS    0x10
```

Tato instrukce slouží k nastavení podpisu $\sigma_{RA} = g_1^{\frac{1}{H(m_r||ID)+sk_{RA}}}$, který je spočten revokační autoritou. Také je uživateli zaslán revokační atribut m_r, který mu byl revokační autoritou přiřazen. Instrukce je součástí algoritmu `Issue`.

```
#define INS_GET_ATTRIBUTES   0x20
```

Pomocí této instrukce uživatel zašle vydavateli své osobní atributy včetně revokačního atributu m_r. Mezi osobní atributy patří atribut m_1, který dále bude sloužit jako nezveřejněný atribut, a atribut m_2, který bude sloužit jako zveřejněný atribut. Instrukce je součástí algoritmu `Issue`.

```
#define INS_SET_SIGNATURES              0x25
#define INS_SET_SIGNATURES_2            0x26
```

Poslední instrukce spadající pod algoritmus `Issue`. Pomocí těchto instrukcí jsou uživateli předány podpisy na atributy. Zejména se jedná o podpisy $\sigma = g_1^{\frac{1}{x_0+x_1m_1+x_2m_2+x_rm_r}}$, $\sigma_{x_1}=\sigma^{x_1}$, $\sigma_{x_2}=\sigma^{x_2}$ a $\sigma_{x_r}=\sigma^{x_r}$.

```
#define INS_SET_EPOCH_NONCE             0x30
```

Instrukce `INS_SET_EPOCH_NONCE` slouží k nastavení náhodné hodnoty *nonce* a časového údaje *epoch*. Náhodná hodnota *nonce* je využita pro výpočet hashe. Časový údaj *epoch* reprezentuje aktuální datum a je použit pro výpočet revokačního pseudonymu *C*. Instrukce je součástí algoritmu `Show-Verify`.

```
#define INS_GET_PROOF                   0x50
#define INS_GET_PROOF2                  0x51
#define INS_GET_PROOF3                  0x52
```

Nejdůležitější instrukce algoritmu `Show-Verify`, pomocí kterých se vypočítají všechny proměnné atributového schématu. Rovněž se v rámci těchto instrukcí předají všechny proměnné, které ověřovatel potřebuje pro ověření.

```
#define INS_TEST                    0x15
#define INS_TEST2                   0x16
```

Tyto instrukce nejsou pro správnou funkcionalitu nutné, avšak slouží k otestování a porovnání všech vypočtených ověřovacích hodnot, tj. t_{verify}, t_{revoke}, t_{sig}, t_{sigI}, t_{sigII}.

5.2 Struktury

Pro deklaraci většiny využitých proměnných slouží datový typ BYTE. Tento datový typ má velikost jednoho bajtu, tj. 0 až 255 a je definován jako:

```
typedef unsigned char BYTE;
```

Základní algebraickou strukturou využitou v této aplikaci je eliptická křivka BN-256. Doménové parametry eliptické křivky byly předem zadány vedoucím práce. Tato eliptická křivka je definována nad prvočíselným polem $\mathbb{F}_p$ s velikostí 32 bajtů. Pro uložení bodu eliptické křivky je vhodné vytvořit následující strukturu:

```
typedef struct{
  BYTE representation;
  BYTE x[EC_SIZE];
  BYTE y[EC_SIZE];
} ECPoint;
```

Velikost jednoho bodu na eliptické křivce je 65 bajtů. X-ová i y-ová souřadnice bodu má velikost 32 bajtů. Pro reprezentaci bodu slouží `representation` bajt, který určí, zdali se jedná o bod v afinním či projektivním tvaru. V tomto případě je při ukládání bodu křivky hodnota `representation` bajtu nastavena na `0x04`, což specifikuje afinní (x, y) tvar souřadnic. Mezi proměnné, které jsou body na křivce, patří revokační pseudonym C, všechny σ hodnoty a všechny t hodnoty.

Pro operaci násobení bodu křivky celým číslem je vytvořena struktura:

```
typedef struct{
  BYTE zero;
  BYTE ecc_multiplier[EC_SIZE];
} ECC_multiplier;
```

Tato struktura je vytvořena pro potřeby MultOS operace *ECC Scalar Multiplicaton* pro správnou reprezentaci násobitele. Aby tato operace pracovala správně, je nutné nastavit první bajt na nulu. Pro násobení bodu křivky jsou za účelem randomizace využita náhodná celá čísla o velikosti 32 bajtů. Tato čísla jsou uložena do struktury:

```
typedef struct{
  ECC_multiplier rho;
  ECC_multiplier rhov;
  ECC_multiplier rhoi;
  ...
}RandomNum;
```

Jedná se o hodnoty ρ, ρ_v, ρ_i, ρ_{mr}, ρ_m, ρ_{eI}, ρ_{eII}. Každý z těchto náhodných násobitelů má velikost 33 bajtů včetně `zero` bajtu.

5.3 Výpočty hlavních hodnot aplikace

Tato část bude věnována postupu použitému při výpočtech zásadních proměnných aplikace. Výpočty jednotlivých proměnných protokolu jsou pro větší přehlednost členěny do funkcí, které jsou volány v instrukci `INS_GET_PROOF`. Všechny proměnné, které bylo nutno na straně uživatele vypočítat, jsou zobrazeny na obr. 1.7. Podrobný popis všech využitých operací je dostupný v oficiální dokumentaci MultOS [13].

5.3.1 Výpočet revokačních hodnot *i* a *C*

Atributové schéma podporuje revokaci uživatele, proto je nutné vypočítat příslušné hodnoty, aby případná revokace mohla být úspěšně provedena. Počet verifikací uživatele za jeden kalendářní den je omezen na sto, což je důležité před samotným výpočtem revokačních hodnot ošetřit. V průběhu každé verifikace ověřovatel uživateli zasílá aktuální časovou epochu. Proto uživatel během jednotlivých verifikací pomocí funkce `memcmp` porovná epochu uloženou na kartě s aktuální epochou, která mu byla zaslána ověřovatelem. Pokud se epochy neshodují, uživatel přepíše epochu na kartě epochou přijatou od ověřovatele a nastaví počet verifikací na nulu. V případě, kdy se epochy shodují, uživatel inkrementuje aktuální počet verifikací o jedna. Jestliže dojde k překročení hranice sta ověření za jednu epochu, uživatel zašle ověřovateli návratový kód `0x6406`, na který ověřovatel reaguje hláškou upozorňující o překročení verifikačních pokusů.

Po ošetření verifikací lze přistoupit k samotnému výpočtu jednotlivých revokačních hodnot. Pro každou verifikaci uživatele v daném dni je potřeba vypočítat jedinečný revokační pseudonym $C = g_1^{\frac{1}{i-m_r+H(epoch)}}$. Prvním krokem je výpočet hodnoty $i = \alpha_1 e_I + \alpha_2 e_{II}$. Hodnoty použité pro tento výpočet jsou součástí parametrů revokační autority. Pro každou verifikaci je kombinace randomizérů e_I a e_{II} jiná, tím je docíleno požadované jedinečnosti. Unikátní kombinace randomizérů e_I a e_{II} je získána rozložením aktuálního počtu verifikací na desítky a jednotky pomocí operátorů / (tj. dělení) a % (tj. zbytek po dělení). Získané hodnoty jsou uloženy do pomocných proměnných, na jejichž základě se podmínkou `if` zvolí příslušný randomizér a podpis z množiny $\{(e_1, \sigma_{e_1}), \ldots, (e_{10}, \sigma_{e_{10}})\}$, tzn. pokud aktuální počet verifikací je 28, tak $(e_I = e_2, \sigma_{e_I} = \sigma_{e_2})$

a ($e_{II} = e_8$, $\sigma_{e_{II}} = \sigma_{e_8}$). K výpočtu hodnoty i je využit zásobník, do kterého se nedříve vloží 32 bajtové proměnné α_1 a randomizér e_I. Tyto dvě hodnoty se pomocí operace *MultiplyN* s kódem `0x10` vynásobí:

```
__push(BLOCKCAST(EC_SIZE) (alpha1));
__push(BLOCKCAST(EC_SIZE) (eI.ecc_multiplier));
__code(PRIM, 0x10, EC_SIZE);
```

Podobným způsobem je spočítán součin $\alpha_2 e_{II}$. Následně jsou tyto dva získané součiny instrukcí `ADDN` sečteny:

```
__code(ADDN, 2*EC_SIZE);
```

V následujícím kroku se ze zásobníku instrukcí `POPN` vyjme součin $\alpha_2 e_{II}$. Na závěr je výsledný součet $\alpha_1 e_I + \alpha_2 e_{II}$ pomocí instrukce `STORE` uložen do proměnné `i_tmp`:

```
__code(POPN, 2*EC_SIZE);
__code(STORE, i_tmp, 2*EC_SIZE);
```

Před dalším použitím hodnoty i je nezbytné ji zredukovat modulo řád q:

```
ModularReduction(2*EC_SIZE, EC_SIZE, i_tmp, order);
```

Tímto lze přejít k výpočtu revokačního pseudonymu C. K výpočtu je potřeba bod g_1, který je součástí doménových parametrů eliptické křivky, získaná hodnota i, revokační atribut m_r a hash aktuální časové epochy. K vytvoření hashe aktuální epochy byla využita hashovací funkce SHA1. Prvním krokem je výpočet jmenovatele v exponentu. Nejdříve se pomocí instrukce `ADDN` spočte součet $i + H(epoch)$, který je uložen do pomocné proměnné `tmp_sub`. Následně je tento součet funkcí `memcmp` porovnán s revokačním atributem m_r. Pokud je součet větší než revokační atribut m_r, tak se tento atribut instrukcí `SUBN` odečte od proměnné `tmp_sub` a výsledek se instrukcí `STORE` uloží do proměnné `inverse`:

```
__push(BLOCKCAST(EC_SIZE) (tmp_sub));
__push(BLOCKCAST(EC_SIZE) (mr));
__code(SUBN, EC_SIZE);
__code(POPN, EC_SIZE);
__code(STORE, inverse.ecc_multiplier, EC_SIZE);
```

Může však nastat situace, kdy je součet $i + H(epoch)$ menší než revokační atribut m_r. To by v důsledku znamenalo, že pokud by se od součtu odečetla hodnota m_r, tak výsledek by byl záporný. Této situaci se dá předejít úpravou jmenovatele na tvar $q - (m_r - (i + H(epoch)))$. Výpočet je proveden obdobně jako v předchozím případě, tj. s využitím instrukce `SUBN`.

V druhém kroku je provedena modulární inverze vypočteného jmenovatele, který je uložen v proměnné `inverse`. K výpočtu modulární inverze je využita operace *Modular Inverse* s kódem `0xD0` a číslem `1`, které udává, že řád q je prvočíslo. Vstupními parametry operace jsou velikost řádu q, řád q, velikost proměnné `inverse`, proměnná `inverse` a proměnná, do které bude výsledná inverze uložena. Všechny parametry jsou vloženy do zásobníku a pomocí kódu `0xD0` je vykonána modulární inverze:

```
__push(EC_SIZE);
__push(order);
__push(EC_SIZE);
__push((BYTE*)&inverse.ecc_multiplier);
__push((BYTE*)&inverse.ecc_multiplier);
__code(PRIM, 0xD0, 1);
```

Posledním krokem je vynásobení generátoru g_1 hodnotou v proměnné `inverse`. K tomuto účelu je využita operace *ECC Scalar Multiplicaton* s kódem `0xD4`. Vstupem operace jsou doménové parametry křivky, bod g_1, 33 bajtový násobitel `inverse` a proměnná `C`, do které bude spočtený revokační pseudonym uložen:

```
__push (ecDomainParams);
__push ((BYTE*)&pointG);
__push ((BYTE*)&inverse);
__push ((BYTE*)&C);
__code (PRIM,0xD4)
```

5.3.2 Generování náhodných hodnot

Atributové schéma je založeno na randomizaci, proto je důležité vygenerovat patřičné náhodné hodnoty. Pro vygenerování slouží operace *Get Random Number* s kódem `0xC4`. Jejím výstupem je 8bajtové náhodné číslo uložené v zásobníku. Pro potřeby atributového schématu je nutno vygenerovat náhodná čísla o velikosti řádu q, tedy 32 bajtů. Proto je tato operace provedena čtyřikrát. Jako příklad poslouží vygenerování náhodné hodnoty ρ:

```
__code(PRIM, 0xc4);
__code(PRIM, 0xc4);
__code(PRIM, 0xc4);
__code(PRIM, 0xc4);
__code(STORE,rndnum.rho.ecc_multiplier, EC_SIZE);
```

Vygenerované náhodné číslo je ze zásobníku instrukcí `STORE` uloženo do příslušné proměnné struktury `RandomNum`. Postup pro generování hodnot ρ_v, ρ_i, ρ_{mr}, ρ_m, ρ_{eI}, ρ_{eII} je obdobný.

5.3.3 Výpočet σ hodnot

První ze σ hodnot, kterou je potřeba vypočítat, je $\hat{\sigma} = \sigma^{\rho}$. Výpočet je proveden pomocí operace *ECC Scalar Multiplicaton* s kódem `0xD4`, kde vstupem operace jsou doménové parametry, vydavatelem vypočítaný podpis na atributy σ, 33 bajtový náhodný násobitel ρ a proměnná `sigmaroof`, do které bude spočtený randomizovaný podpis $\hat{\sigma}$ uložen. Princip výpočtu je opět založen na vkládání příslušných hodnot do zásobníku, kde se vykoná požadovaná operace s kódem `0xD4`:

```
__push(ecDomainParams);
__push((BYTE*)&sigma);
__push ((BYTE*)&rndnum.rho);
__push ((BYTE*)&sigmaroof);
__code(PRIM, 0xD4);
```

Dalšími σ hodnotami, které je potřeba randomizovat, jsou hodnoty $\hat{\sigma}_{e_I}=\sigma^{\rho}_{e_I}$ a $\hat{\sigma}_{e_{II}}=\sigma^{\rho}_{e_{II}}$. Tyto hodnoty slouží jako důkaz, že pro výpočet revokačního pseudonymu C byly využity platné hodnoty. Podpisy σ_{e_I} a $\sigma_{e_{II}}$ jsou součástí parametrů, které vygenerovala revokační autorita, a jejich randomizace je založena na stejném principu jako u podpisu $\hat{\sigma}$.

Posledními σ hodnotami jsou $\bar{\sigma}_{e_I}=\hat{\sigma}_{e_I}^{-e_I} g_1^{\rho}$ a $\bar{\sigma}_{e_{II}}=\hat{\sigma}_{e_{II}}^{-e_{II}} g_1^{\rho}$, jejichž výpočet se oproti předchozím hodnotám skládá z více kroků. Pro příklad poslouží výpočet první zmiňované hodnoty. V prvním kroku je využita operace *ECC Scalar Multiplicaton*, se kterou se vypočítá hodnota $\hat{\sigma}_{e_I}^{e_I}$. Vypočtená hodnota se uloží do proměnné `sigmadheI`. Jelikož exponent e_I je záporný, tak dalším krokem je výpočet inverze bodu na eliptické křivce (tj. inverze hodnoty $\hat{\sigma}_{e_I}^{e_I}$). Tento krok je realizován pomocí operace *ECC Inverse* s kódem `0xD3`. Vstupem funkce jsou doménové parametry, bod na eliptické křivce určený k inverzi a proměnná, do které se výsledná inverze uloží:

```
__push (ecDomainParams);
__push ((BYTE*)&sigmadheI);
__push ((BYTE*)&sigmadheI);
__code (PRIM,0xD3);
```

Tímto krokem byl získán bod $\hat{\sigma}_{e_I}^{-e_I}$. Následuje krok, kdy je do proměnné `tmp1_g1` uložen randomizovaný bod g_1^{ρ}, který byl vypočítán operací *ECC Scalar Multiplicaton*. Na závěr dojde k sečtení těchto dvou získaných hodnot. Sčítání bodů na eliptické křivce provádí operace *ECC*

Addition s kódem `0xD0`. Vstupními hodnotami operace jsou doménové parametry, body na eliptické křivce určené pro součet a proměnná, do které se výsledný součet uloží:

```
__push (ecDomainParams);
__push ((BYTE*)&sigmadheI);
__push ((BYTE*)&tmp1_g1);
__push ((BYTE*)&sigmadheI);
__code (PRIM, 0xD0);
```

5.3.4 Výpočet *t* hodnot a hashe *e*

Všechny *t* hodnoty jsou body na eliptické křivce a slouží zejména pro finální ověření správné funkcionality. Mezi *t* hodnoty patří t_{verify}, t_{revoke}, t_{sig}, t_{sigI}, t_{sigII}. Výpočet těchto hodnot sestává pouze ze dvou operací, a to ze sčítání a násobení bodů na eliptické křivce. K tomuto jsou využity operace *ECC Addition* a *ECC Scalar Multiplicaton.* Jelikož princip výpočtů je u každé hodnoty podobný, bude zde podrobně popsán postup pouze u hodnoty t_{verify}, jejíž výpočet je ze všech hodnot nejkomplexnější.

Výpočet je rozdělen do pěti kroků. První krok je zaměřen na výpočet hodnoty $g_1^{\rho_v}$, který je proveden s využitím operace *ECC Scalar Multiplicaton*. Do zásobníku se vloží doménové parametry, bod g_1, náhodný násobitel ρ_v a proměnná `tmp3_g1`, do které bude uložen výsledek:

```
__push (ecDomainParams);
__push ((BYTE*)&pointG);
__push ((BYTE*)&rndnum.rhov);
__push ((BYTE*)&tmp3_g1);
__code (PRIM,0xD4);
```

Druhým krokem je výpočet hodnoty $\sigma_{x_r}^{\rho_{m_r} \cdot \rho}$. Tento krok je rozdělen do dvou dílčích částí. V první části se pomocí *ECC Scalar Multiplicaton* spočítá $\sigma_{x_r}^{\rho_{m_r}}$. Vstupem operace jsou doménové parametry, podpis na revokační atribut σ_{x_r}, náhodný násobitel ρ_{m_r} a proměnná `tmp1_tver`, do které bude uložen výsledek:

```
__push (ecDomainParams);
__push ((BYTE*)&sigmaxr);
__push ((BYTE*)&rndnum.rhomr);
__push ((BYTE*)&tmp1_tver);
__code (PRIM,0xD4);
```

V druhé části tohoto kroku se získaná proměnná `tmp1_tver` s hodnotou $\sigma_{x_r}^{\rho_{m_r}}$ opět pomocí *ECC Scalar Multiplicaton* vynásobí náhodným násobitelem ρ:

```
__push (ecDomainParams);
__push ((BYTE*)&tmp1_tver);
__push ((BYTE*)&rndnum.rho);
__push ((BYTE*)&tmp1_tver);
__code (PRIM,0xD4);
```

Vykonáním této operace se obsah proměnné `tmp1_tver` změní na $\sigma_{x_r}^{\rho_{m_r} \cdot \rho}$.

Ve třetím kroku se provede součet dosud spočtených hodnot, tj. $g_1^{\rho v}$ a $\sigma_{x_r}^{\rho_{m_r} \cdot \rho}$. Tohoto součtu je docíleno pomocí operace *ECC Addition*. Vstupem jsou doménové parametry, hodnota $g_1^{\rho v}$ uložená v proměnné `tmp3_g1` , hodnota $\sigma_{x_r}^{\rho_{m_r} \cdot \rho}$ uložená v proměnné `tmp1_tver` a proměnná `tverify`, do které se zapíše výsledný součet:

```
__push (ecDomainParams);
__push ((BYTE*)&tmp3_g1);
__push ((BYTE*)&tmp1_tver);
__push ((BYTE*)&tverify);
__code (PRIM, 0xD0);
```

Čtvrtý krok se věnuje výpočtu proměnné `tmp2_tver`, která bude reprezentovat hodnotu $\sigma_{x_1}^{\rho_m \cdot \rho}$. Podobně jak u proměnné `tmp1_tver` je i zde výpočet rozdělen do dvou částí. V první části se pomocí *ECC Scalar Multiplication* spočte $\sigma_{x_1}^{\rho_m}$. Do zásobníku se vloží doménové parametry, podpis pro nezveřejněný atribut σ_{x_1} a náhodný násobitel ρ_m:

```
__push (ecDomainParams);
__push ((BYTE*)&sigmax1);
__push ((BYTE*)&rndnum.rhom);
__push ((BYTE*)&tmp2_tver);
__code (PRIM,0xD4);
```

V druhé části se získaná proměnná `tmp2_tver` s hodnotou $\sigma_{x_1}^{\rho_m}$ vynásobí pomocí *ECC Scalar Multiplicaton* náhodným násobitelem ρ:

```
__push (ecDomainParams);
__push ((BYTE*)&tmp2_tver);
__push ((BYTE*)&rndnum.rho);
```

```
__push ((BYTE*)&tmp2_tver);
__code (PRIM,0xD4);
```

V závěrečném pátém kroku se provede finální součet získaných hodnot. Sčítat se budou proměnná `tverify`, ve které je uložena hodnota $g_1^{\rho_v} \sigma_{x_r}^{\rho_{m_r} \cdot \rho}$ a proměnná `tmp2_tver` s hodnotou $\sigma_{x_1}^{\rho_m \cdot \rho}$. Součet se provede pomocí *ECC Addition*:

```
__push (ecDomainParams);
__push ((BYTE*)&tverify);
__push ((BYTE*)&tmp2_tver);
__push ((BYTE*)&tverify);
__code (PRIM, 0xD0);
```

Po provedení této operace je v proměnné `tverify` uložena finální podoba hodnoty t_{verify}, tj. $g_1^{\rho_v} \sigma_{x_r}^{\rho_{m_r} \cdot \rho} \sigma_{x_1}^{\rho_m \cdot \rho}$. S využitím odpovídajících vstupních hodnot, byly obdobným principem vypočítány všechny ostatní *t* hodnoty.

Ze získaných *t* hodnot, σ hodnot, revokačního pseudonymu *C* a náhodné hodnoty *nonce* je vytvořen hash, který je uložen do proměnné `e`. Všechny zmíněné hodnoty jsou postupně ukládány do pomocné proměnné `hashdata`, která slouží jako vstup hashovací funkce. Jako hashovací funkce byla zvolena SHA-256 s velikostí hashe 32bajtů. Výsledný hash je využit pro výpočty hodnot s_m, s_{m_r}, s_v, s_i, s_{e_I}, $s_{e_{II}}$.

5.3.5 Výpočet proměnné π

Proměnná π se skládá z hodnot s_m, s_{m_r}, s_v, s_i, s_{e_I}, $s_{e_{II}}$, které jsou ověřovatelem využity pro výpočty *t* hodnot. Výpočty jednotlivých hodnot se skládají ze dvou operací. První operací je násobení, které je obsaženo ve výpočtech všech těchto hodnot. Druhou operací je buď sčítání, nebo odčítání. V této části bude popsán princip výpočtu jedné hodnoty s operací sčítání a jedné s operací odčítání.

Pro příklad, kdy je využita operace sčítání, poslouží výpočet hodnoty $s_v = \rho_v + e\rho$. Každá ze vstupních hodnot (ρ_v, e, ρ) má velikost 32 bajtů. Pro operaci sčítání je nutné, aby obě sčítané hodnoty měly stejnou bajtovou velikost. To však neplatí, jelikož součin $e\rho$ má velikost 64 bajtů a hodnota ρ_v jen 32 bajtů. Z toho důvodu se do zásobníku vloží instrukcí `PUSHZ` 32 bajtů nul. To zajistí, že následně vložená hodnota ρ_v bude mít velikost 64 bajtů:

```
__code(PUSHZ, EC_SIZE);
__push(BLOCKCAST(EC_SIZE) (rndnum.rhov.ecc_multiplier));
```

Poté se do zásobníku vloží hash *e* a náhodné číslo ρ. Pro vynásobení těchto hodnot je využita operace *MultiplyN* s kódem `0x10`:

```
__push(BLOCKCAST(EC_SIZE) (e));
__push(BLOCKCAST(EC_SIZE) (rndnum.rho.ecc_multiplier));
```

```
__code(PRIM, 0x10, EC_SIZE);
```

Takto získané 64 bajtové hodnoty ρ_v a $e\rho$ již lze sečíst pomocí instrukce `ADDN`:

```
__code(ADDN, 2*EC_SIZE);
```

Následně je potřeba ze zásobníku instrukcí `POPN` vyjmout součin $e\rho$. Nakonec je výsledný součet instrukcí `STORE` uložen do proměnné `tmp_sv`:

```
__code(POPN, 2*EC_SIZE);
__code(STORE, tmp_sv, 2*EC_SIZE);
```

Výslednou hodnotu v proměnné `tmp_sv` je nutné zredukovat modulo řád q:

```
ModularReduction(2*EC_SIZE, EC_SIZE, tmp_sv, order);
```

Hodnota s_i, která také využívá operaci sčítání, je vypočítána obdobným způsobem.

Druhým příkladem bude výpočet hodnoty $s_m = \rho_m - em_1$, ve kterém je využita operace odčítání. U této operace je důležité zohlednit fakt, že od 32 bajtové hodnoty ρ_m je odečítán součin em_1, jehož velikost je 64 bajtů. Jelikož nelze počítat v záporných hodnotách, je potřeba upravit výpočet původní hodnoty na $s_m = q - (em_1 - \rho_m)$. Prvním krokem je výpočet součinu em_1. Do zásobníku se vloží hodnoty hashe e a skrytého atributu m_1, které se operací *MultiplyN* vynásobí:

```
__push(BLOCKCAST(EC_SIZE) (e));
__push(BLOCKCAST(EC_SIZE) (m1));
__code(PRIM, 0x10, EC_SIZE);
```

Následně se proměnná ρ_m doplní pomocí nul na velikost 64 bajtů tak, aby šlo příslušné hodnoty od sebe odečíst:

```
__code(PUSHZ, EC_SIZE);
__push(BLOCKCAST(EC_SIZE) (rndnum.rhom.ecc_multiplier));
```

Poté, co jsou obě hodnoty stejně velké, mohou se od sebe pomocí instrukce `SUBN` odečíst:

```
__code(SUBN, 2*EC_SIZE);
```

Po provedení odčítání se ze zásobníku pomocí instrukce `POPN` vyjme součin em_1 a výsledek se instrukcí `STORE` uloží do proměnné `tmp_sm`:

```
__code(POPN, 2*EC_SIZE) ;
__code(STORE, tmp_sm, 2*EC_SIZE);
```

Tímto je v proměnné `tmp_sm` uložena hodnota $em_1 - \rho_m$, kterou je potřeba zredukovat:

```
ModularReduction(2*EC_SIZE, EC_SIZE, tmp_sm, order);
```

V posledním kroku se s využitím funkce `memcpy` zkopíruje zredukovaná hodnota do proměnné `sm`, která se následně pomocí instrukce `SUBN` odečte od řádu q:

```
__push(BLOCKCAST(EC_SIZE)(ecDomainParams+EC_DomainParams_N));
__push(BLOCKCAST(EC_SIZE) (sm));
__code(SUBN, EC_SIZE);
```

Instrukcí `POPN` se ze zásobníku vyjme proměnná `sm` (tj. hodnota $em_1 - \rho_m$) a získaný rozdíl je instrukcí `STORE` uložen do proměnné `sm`, jejíž obsah se přepíše na hodnotu $q - (em_1 - \rho_m)$:

```
__code(POPN, EC_SIZE) ;
__code(STORE, sm, EC_SIZE);
```

Obdobným způsobem jsou spočítány všechny ostatní hodnoty využívající operaci odčítání, tj. s_{m_r}, s_{e_I}, $s_{e_{II}}$. Všechny hodnoty zmíněné v této podkapitole byly postupně ukládány do proměnné `phi`, která je společně se zveřejněným atributem m_2 instrukcí `INS_GET_PROOF3` zaslána ověřovateli.

5.4 Nahrání a otestování aplikace

Všechny hodnoty potřebné k ověření se uloží do proměnné `apdu_data`, která je deklarována ve veřejné paměti. V rámci instrukcí `INS_GET_PROOF`, `INS_GET_PROOF2` a `INS_GET_PROOF3` jsou následně zaslány ověřovateli. Pro úspěšnou autentizaci není nutné zasílat *t* hodnoty, avšak lze je využít pro kontrolu, zda se shodují s *t* hodnotami vypočtenými stranou ověřovatele.

Před samotným otestováním je nutno danou aplikaci nahrát na čipovou kartu. K tomuto účelu je využit nástroj MUtil. Pro úspěšné nahrání aplikace je důležité specifikovat velikost RAM paměti. Toho lze docílit pomocí příkazu `hls -t <název_souboru>.hzx`, který vypíše paměťové sekce `.hzx` souboru. Velikost dynamické RAM paměti v decimální i hexadecimální formě je definovaná pod názvem `.DB`, viz obr. 5.1. Aplikace využívá 231 z 1280 dostupných bajtů dynamické RAM paměti, 256 z 529 dostupných bajtů veřejné RAM paměti a 5615 z 18000 dostupných bajtů EEPROM paměti. Zbylá část procesu nahrávání je popsána v podkapitole 4.2.1.

```
C:\temp\demo-workspace\card1\Debug>hls -t card1.hzx
   start     stop     size  decimal  name
00000000 00000c06      c07     3079  .text
00000000 000015ee     15ef     5615  .SB
00000000 000000e6       e7      231  .DB
00000000 000000ff      100      256  .PB
```

Obr. 5.1: Výpis příkazu `hls -t`

Otestování funkčnosti aplikace proběhlo ve spolupráci se studentem, který vytvořil stranu terminálu na jednodeskovém počítači Raspberry PI. Výstup terminálu po vložení čipové karty s autentizační aplikací je rozdělen do dílčích částí, které jsou zobrazeny na obr. 5.2, obr. 5.3 a obr. 5.4.

```
REVOCATION AUTHORITY PART

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! (277ms)

ISSUER PART

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! (713ms)
```

Obr. 5.2: Výstup terminálu – algoritmus `Issue`

První část výpisu je věnována algoritmu `Issue`. V rámci první části algoritmu `Issue` terminál vypisuje přijatý identifikátor uživatele společně s informací o zaslání revokačního atributu m_r a podpisu σ_{RA}. Čas potřebný pro vydání revokačního atributu je 277ms. V druhé části algoritmu `Issue` jsou vypsány všechny kartou využívané atributy včetně informace o zaslání podpisů na atributy σ, σ_{x_1}, σ_{x_2}, σ_{x_r}. Čas potřebný pro vydání podpisů na atributy je 713ms.

```
VERIFICATION

Printing received data:
Sigma_roof_x:           22 20 a1 ce f6 44 03 49 80 ee 4a 85 33 25 20 9b 67 d9 ce 29 64 97 5a e5 2a f3 2d 18 2b c7 64 8b
Sigma_roof_y:           1d 3d 92 88 41 12 75 91 4a ea d7 c7 f9 9b 37 bd 62 d5 c4 22 2e 54 fb 09 89 1d 2d 0f 05 1f 6b 67
Sigma_roof_ei_x:        1c 15 24 39 43 7a e8 fc 5d 25 64 1a 12 5d 39 1e 98 cf 27 5d f1 d3 d7 c5 a1 8f 6d b9 ba 12 e8 0c
Sigma_roof_ei_y:        21 64 b9 2d c3 4d 83 73 f0 ac a9 60 6f 57 08 d4 0a 97 8c 99 62 cf 28 1f e6 b7 ed 6f 7d 11 4f 14
Sigma_roof_eii_x:       12 85 c7 90 b3 50 ca 0b 4e ab 5f 58 f5 aa 4d 37 5a 5e 77 2e 00 a6 c3 69 35 33 69 2a c9 26 f0 a5
Sigma_roof_eii_y:       0e 03 39 b3 d9 23 ec 0b c9 66 e8 a9 2a 90 86 77 db dd 41 02 fd 31 1f 40 6b b2 f4 e5 6f 7a 6d 91
C_x:                    23 36 9a ae 4e 3b 0b e9 9e 2f 96 0c 53 fd dd 63 6a 36 66 de 3f 20 7c cd 49 7f a5 a6 6b 84 81 3f
C_y:                    24 59 28 0f 4f ff 20 de 9a fb ee 70 57 77 b3 b7 6a e8 d0 66 27 6c 11 0c 0d 9e c5 67 c3 16 17 fe
Sigma_neg_ei_x:         08 78 f6 9f 5f e5 a8 73 bd 60 96 a1 fa d2 98 8f 41 ac a3 fc 41 c4 c4 06 59 1b 55 2b 53 2c cc 33
Sigma_neg_ei_y:         12 6e 7d c8 16 ba 60 c3 8a c6 a7 84 90 4a 2b 0f 35 24 21 4b 04 b3 11 9a a8 82 f2 8e 30 92 b4 6e
Sigma_neg_eii_x:        1c 38 2b 03 27 0e d3 7b 7e 12 e4 1c b3 dd aa d2 3a 27 b5 d6 05 c5 1d 96 eb 0a 69 10 41 b9 6e 28
Sigma_neg_eii_y:        20 65 3e cf d0 ac 8c fb ef a8 97 87 40 5a df a2 4f 3a cf 64 76 11 00 ee 1f 5c 15 ae c9 dd d3 d4
e:                      e3 48 0f 1a b2 db a6 9b a9 ff 06 89 94 37 bd 8a 83 90 d4 16 db 46 71 f9 5d e5 e6 63 71 94 59 09
sm1:                    17 11 17 a5 60 17 98 84 f3 b1 ad 8d 97 4f 46 8c ea 26 e4 80 25 84 d0 1a 08 f3 33 2f 1d c3 e3 bf
sv:                     0d f0 6d a1 9a 09 1c ac 6f 50 7c 64 7c a2 73 e9 89 1b 8b 99 18 84 b2 a4 31 76 66 31 1d cc 7f 0d
smr:                    11 38 7c c3 97 d5 27 f4 66 73 c0 8e f7 8c d8 78 ee 25 5e 03 b4 78 10 f5 6d 2b f8 0e 59 d1 8e e4
si:                     0f 52 87 f0 a6 44 11 7e a7 13 0b 23 f6 f1 e0 73 48 6b 76 c8 14 b4 14 c1 0f 2d 08 c8 bb 94 2f bf
sei:                    22 c4 f7 14 88 0e a0 e5 24 55 b8 ff 7d 17 ad 28 40 45 d2 6a 74 01 32 0e 7e 5e 6b ea 0d ba 02 73
seii:                   0e b7 33 d4 14 85 42 23 89 41 0d fb 5f 3d 97 69 57 be 38 94 85 32 81 04 86 82 f3 fd ee 54 22 44
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:  e3 48 0f 1a b2 db a6 9b a9 ff 06 89 94 37 bd 8a 83 90 d4 16 db 46 71 f9 5d e5 e6 63 71 94 59 09
hash_user:      e3 48 0f 1a b2 db a6 9b a9 ff 06 89 94 37 bd 8a 83 90 d4 16 db 46 71 f9 5d e5 e6 63 71 94 59 09
```

Obr. 5.3: Výstup terminálu – algoritmus `Show-Verify`

V druhé části výpisu jsou vypsány všechny hodnoty přijaté v rámci instrukcí `INS_GET_PROOF`, `INS_GET_PROOF2`, `INS_GET_PROOF3`, které ověřovatel potřebuje k výpočtu ověření, tj. t hodnot. Jedná se o hodnoty $\hat{\sigma}$, $\hat{\sigma}_{e_I}$, $\hat{\sigma}_{e_{II}}$, C, $\bar{\sigma}_{e_I}$, $\bar{\sigma}_{e_{II}}$, e, π a zveřejněný atribut m_2. Následně je porovnán hash vypočtený ověřovatelem s hashem vypočteným uživatelem. Pokud se výsledné hashe shodují, tak ověření proběhlo úspěšně.

```
Comparing T points for debugging
t_verify_card:
x:      03 5e 9a b9 80 2d 58 82 17 12 83 03 c1 1a 8e af ed e8 b0 13 83 78 a6 71 68 8a 68 ba 33 0b ca ff
y:      01 34 0a 28 47 02 63 2d cb 21 27 b0 e7 e3 cb cf 5b ae 98 30 95 f9 77 20 99 c7 05 ba 66 c6 64 6c
t_verify_verifier:
x:      03 5e 9a b9 80 2d 58 82 17 12 83 03 c1 1a 8e af ed e8 b0 13 83 78 a6 71 68 8a 68 ba 33 0b ca ff
y:      01 34 0a 28 47 02 63 2d cb 21 27 b0 e7 e3 cb cf 5b ae 98 30 95 f9 77 20 99 c7 05 ba 66 c6 64 6c
t_revoke_card:
x:      0d 54 d9 4d 32 f2 9a 32 2d 24 31 1d 84 c7 a2 f0 bb f3 a5 40 a9 2b 67 4c 15 f8 fa 22 19 9f 41 dc
y:      01 e5 4c 8e 40 ba cf f5 fe 5d fb f9 20 4b d8 a5 f2 7c 04 8f c8 81 9a bb 8a de a2 76 10 9c 9b 8b
t_revoke_verifier:
x:      0d 54 d9 4d 32 f2 9a 32 2d 24 31 1d 84 c7 a2 f0 bb f3 a5 40 a9 2b 67 4c 15 f8 fa 22 19 9f 41 dc
y:      01 e5 4c 8e 40 ba cf f5 fe 5d fb f9 20 4b d8 a5 f2 7c 04 8f c8 81 9a bb 8a de a2 76 10 9c 9b 8b
ei:     68 bd 76 35 59 0b a1 22 5d 17 a5 6c 07 a9 15 1b 24 73 59 8b 13 a9 dc 64 32 8b cd bd f5 1a 06 5d
eii:    e2 9e 41 6a d2 94 50 1e 8c a2 71 2d 09 df f3 e3 a6 44 d0 16 71 22 b8 29 41 18 d7 7a 7e e2 8b 60
t_sig_card:
x:      12 a6 06 f1 e3 2a 9c b1 ae f7 d0 66 7f e8 f7 ca 40 4a fa c8 20 42 7b 2d 65 e8 bb 3d 04 7e 08 57
y:      1c 56 71 3b e4 a7 48 43 90 87 d6 09 56 9f e8 da 8f 1c 72 81 4e d7 70 2e ab ef 8c 63 28 58 99 6f
t_sig_verifier:
x:      12 a6 06 f1 e3 2a 9c b1 ae f7 d0 66 7f e8 f7 ca 40 4a fa c8 20 42 7b 2d 65 e8 bb 3d 04 7e 08 57
y:      1c 56 71 3b e4 a7 48 43 90 87 d6 09 56 9f e8 da 8f 1c 72 81 4e d7 70 2e ab ef 8c 63 28 58 99 6f
t_sigi_card:
x:      20 0e 65 16 69 30 aa 9d 57 27 c4 a1 bc 1c 53 ce 1b ce 84 ce 0b eb 0a 08 13 a2 ce 0c b1 4b 9d 8e
y:      04 c9 ee c8 6e 6d e7 cc 82 2c ad dc 26 6b 3e ff ca ff a9 25 2f fa f6 8b 08 f0 a7 ad 0e fc 3a 00
t_sigi_verifier:
x:      20 0e 65 16 69 30 aa 9d 57 27 c4 a1 bc 1c 53 ce 1b ce 84 ce 0b eb 0a 08 13 a2 ce 0c b1 4b 9d 8e
y:      04 c9 ee c8 6e 6d e7 cc 82 2c ad dc 26 6b 3e ff ca ff a9 25 2f fa f6 8b 08 f0 a7 ad 0e fc 3a 00
t_sigii_card:
x:      13 e5 6e 93 a7 33 ad fd 10 d5 06 68 7c 70 51 57 e1 12 d0 9c 17 12 aa bf 2b 74 0a a0 56 7c 3b d6
y:      23 61 70 13 3d ff f0 66 f6 5e c7 40 28 43 e1 3c 4b 36 f2 b6 a8 e5 bc 62 26 72 62 a9 65 f9 19 86
t_sigii_verifier:
x:      13 e5 6e 93 a7 33 ad fd 10 d5 06 68 7c 70 51 57 e1 12 d0 9c 17 12 aa bf 2b 74 0a a0 56 7c 3b d6
y:      23 61 70 13 3d ff f0 66 f6 5e c7 40 28 43 e1 3c 4b 36 f2 b6 a8 e5 bc 62 26 72 62 a9 65 f9 19 86

Verification successful! (3714ms)
RUN SUCCESSFUL (total time: 5s)
```

Obr. 5.4: Výstup terminálu – porovnání *t* hodnot

Na konci výpisu jsou pro kontrolu porovnány *t* hodnoty vypočtené uživatelem s *t* hodnotami vypočtenými ověřovatelem. V závěru je vypsaná hláška o úspěšné či neúspěšné verifikaci. Ze sta ověření byl stanoven průměrný čas potřebný pro jedno ověření, který činí 3700ms.

5.4.1 Možnosti optimalizace

V současném stavu aplikace podporuje dva atributy, jeden skrytý a jeden odhalený. Z hlediska optimalizace by bylo vhodné rozšířit aplikaci na kartě i terminálu o možnost autentizace pomocí více atributů. Na straně uživatele by se jednalo zejména o úpravu rovnice t_{verify} na podobu $t_{verify} = g_1^{\rho_v} \sigma_{x_r}^{\rho_{m_r} \cdot \rho} \prod_{z \notin D} \sigma_{x_z}^{\rho_{m_z} \cdot \rho}$, kde D reprezentuje množinu všech zveřejněných atributů. Podobným způsobem by bylo potřeba upravit hodnotu s_m na tvar $\langle s_{m_z} = \rho_{m_z} - em_z \rangle_{z \notin D}$. Dalším předmětem optimalizace by mohlo být snížení celkového času ověření. Aplikace byla psána především se zaměřením na správnou funkcionalitu a z důvodu větší přehlednosti je většina dílčích výpočtů jednotlivých hodnot ukládána do pomocných proměnných, které byly využívány při hledání chyb. Odstraněním všech těchto pomocných proměnných by se zmenšila velikost statické paměti a zároveň by se zredukoval počet výskytů funkce `memcpy` (tj. kopírování dat) a tím by se snížil celkový čas ověření. Ke zvýšení efektivity vývoje by mohlo přispět vytvoření knihovny pro jednotlivé operace, které jsou v aplikaci využity. Vytvořením knihovny by šlo předejít vkládání hodnot do zásobníku. Díky tomu by byl vývoj jednodušší a kód přehlednější.

6 ZÁVĚR

Hlavním cílem této bakalářské práce bylo vytvoření aplikace pro atributovou autentizaci s revokací na čipové kartě MultOS, která reprezentuje stranu uživatele. Pro tyto účely byla vytvořena aplikace pro autentizaci pomocí jednoho skrytého a jednoho odhaleného atributu. Aplikace jsou založeny na podpisovém schématu weak Boneh Boyen a algebraickém MAC kódu. Realizované schéma i základní primitiva a protokoly využívané v atributové autentizaci jsou pospány v teoretické části. Pro vývoj aplikací bylo nutné nastudovat možnosti operačního systému MultOS a seznámit se se základními principy nástrojů využívaných pro vývoj a správu aplikací. Popis operačního systému MultOS a nástrojů pro vývoj aplikací je rovněž uveden v teoretické části. Při studii vývoje aplikací na platformě MultOS bylo čerpáno především z ukázkových aplikací a dokumentace dostupné na oficiálních stránkách MultOS.

Funkčnost výsledné aplikace nahrané na čipové kartě MultOS byla otestována na straně ověřovatele, tj. terminálu. Stranu ověřovatele vytvořil student Jan Hlinka, se kterým byla při tvorbě praktické části této práce navázána úzká spolupráce. Během řešení praktické části bylo nutné se potýkat s množstvím problémů, které většinou souvisely s výpočtem hlavních proměnných aplikace. Z důvodu nedostatečné podpory ladění aplikací bylo potřeba tyto chyby hledat na základě vzájemného přepočítávání jednotlivých proměnných. Tímto postupem byly všechny příčiny chyb odhaleny a opraveny, a tudíž aplikace zajistí úspěšnou autentizaci s možností revokace.

POUŽITÁ LITERATURA

[1] HAJNÝ, Jan a kol. Technická zpráva k projektu TAČR TL02000398. 2019 [cit. 2020-5-30].

[2] MENEZES, Alfred, Paul C. VAN OORSCHOT a Scott A. VANSTONE. *Handbook of applied cryptography* [online]. Boca Raton: CRC Press, c1997 [cit. 2020-5-30]. Discrete mathematics and its applications. ISBN 0-8493-8523-7. Dostupné z : https://doc.lagout.org/network/3_Cryptography/CRC%20Press%20-%20Handbook%20of%20applied%20Cryptography.pdf

[3] DAMGÅRD, Ivan a Jesper Buus NIELSEN. Commitment Schemes and Zero-Knowledge Protocols [online]. Aarhus University, 2011 [cit. 2020-05-30]. Dostupné z: https://homepages.cwi.nl/~schaffne/courses/crypto/2014/papers/ComZK08.pdf

[4] SCHNORR, C.P. *Efficient Signature Generation by Smart Cards* [online]. Germany: Journal of Cryptology, 1991 [cit. 2020-5-30]. doi:10.1007/BF00196725. Dostupné z: https://link.springer.com/content/pdf/10.1007/BF00196725.pdf

[5] JAO, David a Kayo YOSHIDA. *Boneh-Boyen signatures and the Strong Diffie-Hellman problem* [online]. In: Canada [cit. 2020-5-30]. Dostupné z: https://eprint.iacr.org/2009/221.pdf

[6] CHASE, Melissa, Sarah MEIKLEJOHN a Greg ZAVERUCHA. *Algebraic MACs and Keyed-Verification Anonymous Credentials* [online]. In: ACM CCS 2014 [online]. 2013. [cit. 2020-5-30]. Dostupné z: https://smeiklej.com/files/ccs14.pdf

[7] MAOSCO Limited *MULTOS Developer's Guide* [online]. In: MAO-DOC-TEC-005 v1.43. 2019 [cit. 2020-5-30]. Dostupné z: https://www.multos.com/uploads/MDG.pdf

[8] ISO/IEC 7816-3. *Cards with contacts — Electrical interface and transmission protocols*. 3rd edition. Switzerland: ISO/IEC, 2006. Dostupné také z: http://read.pudn.com/downloads132/doc/comm/563504/ISO-IEC%207816/ISO%2BIEC%207816-3-2006.pdf

[9] EFTLAB *Complete list of APDU responses* [online]. 2019 [cit. 2020-5-30]. Dostupné z: https://www.eftlab.com/knowledge-base/complete-list-of-apdu-responses/

[10] RANKL, Wolfgang a Wolfgang EFFING. *Smart Card Handbook: Third Edition* [online]. In: Munich: Wiley, 2003 [cit. 2020-5-30]. ISBN 0-470-85668-8. Dostupné z: http://index-of.co.uk/Etc/Smart%20Card%20Handbook.pdf

[11] MAOSCO Limited *Guide to Generating Application Load Units* [online]. In: MAO-DOC-TEC-009 v2.9. 2017 [cit. 2020-5-30]. Dostupné z: https://www.multos.com/uploads/GALU.pdf

[12] MAOSCO Limited *Guide to Loading and Deleting* [online]. In: MAO-DOC-TEC-008 v2.29. 2019 [cit. 2020-5-30]. Dostupné z: https://www.multos.com/uploads/GLDA.pdf

[13] MAOSCO Limited *MULTOS Developer's Reference Manual* [online]. In: MAO-DOC-TEC-006 v1.56. 2019 [cit. 2020-5-30]. Dostupné z: https://www.multos.com/uploads/MDRM.pdf

[14] MAOSCO Limited *MULTOS SmartDeck v3.2.1.0 Developer's Reference Manual* [online]. In: 2019 [cit. 2020-5-30]. Dostupné z: https://www.multos.com/uploads/smartdeck-manual.pdf

[15] MULTOS *Tools and SDK* [online]. 2019 [cit. 2020-5-30]. Dostupné z: https://www.multos.com/developer_centre/tools_and_sdk

[16] MAOSCO Limited *MULTOS Utility Manual* [online]. In: MAO-DOC-TEC-017 v2.10.0. 2018 [cit. 2020-5-30]. Dostupné z: https://www.multos.com/uploads/MUM.pdf

SEZNAM POUŽITÝCH ZKRATEK, VELIČIN A SYMBOLŮ

AAM	Application Abstract Machine
ALC	Application Load Certificate
ADC	Application Delete Certificate
APDU	Application Protocol Data Unit
ATR	Answer To Reset
EEPROM	Electrically Erasable Programmable Read-Only Memory
ICC	Integrated Circuit Card
KMA	Key Management Authority
KTU	Key Transformation Unit
MAC	Message Authentication Code
MEL	MultOS Executable Language
RAM	Random Access Memory
ROM	Read Only Memory
TPDU	Transmission Protocol Data Unit
wBB	weak Boneh Boyen

SEZNAM PŘÍLOH

Archív `eloyalty_card.zip`, který obsahuje autentizační aplikaci `card1`.