VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2021 Bc. Šárka Chwastková

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

WEBOVÁ VIZUALIZACE A DEMONSTRÁTOR ANONYMNÍCH POVĚŘENÍ
WEB VISUALIZATION AND DEMONSTRATOR OF ANONYMOUS CREDENTIALS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Šárka Chwastková**
AUTHOR

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

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Studentka: Bc. Šárka Chwastková ***ID:*** 186095
Ročník: 2 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Webová vizualizace a demonstrátor anonymních pověření

POKYNY PRO VYPRACOVÁNÍ:

Nastudujte problematiku atributových pověřovacích schémat a řízení přístupu na základě atributů. Dále nastudujte možnosti vývoje interaktivních webových aplikací a komunikace webového rozhraní s NFC RFID systémy. Na tomto základě vytvořte webovou aplikaci vizualizující princip fungovaní anonymního pověřovacího schématu s revokací. Součástí vyvinuté aplikace bude také demonstrátor funkcionality, který umožní vydání, revokaci a ověření uživatelských pověření uložených na chytrém telefonu s podporou technologie NFC.

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] HAJNÝ, J.; DZURENDA, P.; CAMENISCH, J.; DRIJVERS, M. Fast Keyed-Verification Anonymous Credentials on Standard Smart Cards. In ICT Systems Security and Privacy Protection. Springer Nature Switzerland, 2019. s. 1-13. ISBN: 978-3-030-22312-0.

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

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

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

UPOZORNĚNÍ:
Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být 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.

ABSTRAKT

Tato práce se zabývá tématem atributové autentizace s revokovatelnými anonymními pověřeními. Hlavním zaměřením této práce je implementace tohoto schématu prostřednictvím webové aplikace. Webová aplikace slouží primárně jako vizualizace, která pomocí animací seznámí uživatele s fungováním tohoto schématu, a také jako praktický demonstrátor. Data a kryptografické výpočty pro jednotlivé protokoly systému zajišťuje poskytnutá kryptografická C aplikace, která komunikuje s vytvořenou aplikací. Webová aplikace je také schopna prostřednictvím prohlížeče komunikovat s připojenou čtečkou čipových karet a čipovou kartou MultOS a zajistit tak přenos APDU příkazů a odpovědí mezi čipovou kartou a poskytnutou C aplikací.

KLÍČOVÁ SLOVA

anonymní pověření s revokací, atributové pověření, čipová karta, autentizace, anonymizace, vizualizace, demonstrátor, webová aplikace, revokace

ABSTRACT

This thesis deals with the topic of attribute based credentials with revocable anonymous credentials. The main focus of this work is the implementation of this scheme through a web application. The web application serves primarily as a visualization, which shows the functionality of this scheme through animations, and also as a practical demonstrator. Data and cryptographic calculations for individual system protocols are provided by the given cryptographic C application that communicates with the created application. The web application is also able to communicate with the connected smart card reader and the MultOS smart card and thus create the transmission of APDU commands and responses between the smart card and provided C application.

KEYWORDS

revocable anonymous credential, attributes-based credential, smart card, authentication, anonymization, visualization, demonstrator, web application, revocation

CHWASTKOVÁ, Šárka. *Webová vizualizace a demonstrátor anonymních pověření*. Brno, 2021, 72 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Petr Dzurenda, Ph.D.

Vysázeno pomocí balíčku `thesis` verze 4.00; `http://latex.feec.vutbr.cz`

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Webová vizualizace a demonstrátor anonymních pověření“ jsem vypracovala samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autorka uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušila autorská práva třetích osob, zejména jsem nezasáhla nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědoma 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 autorky

PODĚKOVÁNÍ

Rád bych poděkovala vedoucímu diplomové práce panu Ing. Petru Dzurendovi, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam výpisů

Úvod

V současné době, kdy jsou moderní digitální technologie na výrazném vzestupu a uživatelé zpřístupňují jednotlivým poskytovatelům služeb svá osobní a citlivá data, je důležité, aby tato data byla dostatečně chráněna. K zajištění co nejvíce bezpečného ověřování identity uživatele v přístupových systémech jsou dostupné různé techniky autentizace, jejichž počet stoupá a ty stávající se stále více vyvíjí.

Téma bezpečnosti autentizace se dotýká každého běžného uživatele v moderním světě. Příkladem je přihlašování do internetového bankovnictví nebo jiných internetových služeb, přístup k mobilním sítím nebo technologiím *Internet věcí – Internet of Things* (IoT), věrnostní členství u obchodníků, přístup do budov apod.

Mezi techniky autentizace patří i atributová autentizace a její protokoly. Atributová autentizace klade důraz na to, aby se uživatel pro přístup ke službě prokazoval ověřovateli jen těmi atributy, které jsou při přístupu ke službě nezbytně nutné. Tímto je zajištěna jeho anonymita a minimalizace rizika zneužití jeho identity útočníkem.

Tato práce se zabývá atributovým schématem, které využívá klíčově ověřitelná anonymní pověření s revokací, tzv. Revocable Keyed-Verification Anonymnous Credentials (RKVAC). V rámci praktické části práce je vytvořena webová aplikace, která vizualizuje schéma, zároveň přes rozhraní čtečky karet komunikuje s čipovou kartou, případně s mobilním telefonem s podporou technologie NFC a předinstalovanou kartovou aplikací RKVAC a stejnojmenným kryptografickým jádrem psaném v jazyce C, s funkcemi pro nastavení systému, personalizaci, vydání digitálního pověření na základě atributů, ale i s funkcemi pro ověření těchto pověření a pro revokaci uživatelů.

1 Kryptografické metody a primitiva

Protokoly, jejichž popis je součástí této práce jsou postaveny na následujících základních kryptografických metodách a principech.

1.1 Protokoly s nulovou znalostí

Protokoly s nulovou znalostí jsou jednou z metod, jak se bezpečně autentizovat k elektronické službě [2]. Jsou založené na důkazu nulových znalostí. To znamená, že uživatel, který chce ke službě přistupovat, se prokáže pouze tajnou informací, kterou dokazuje, že má oprávnění, tzn. že zná tajemství nebo ne. Obsah této informace zůstává tajný a ani během procesu ověřování není odhalen. Protokoly nulových znalostí musí zahrnovat tři základní vlastnosti a to:

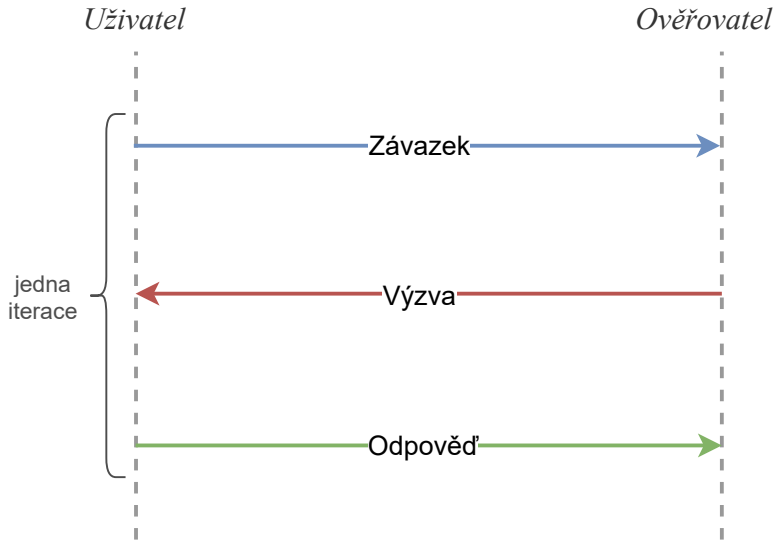
1. **Úplnost** – pokud uživatel zná tajnou informaci, vždy přesvědčí ověřovatele o jeho znalosti. To znamená, že na výzvu, která přichází od ověřovatele, je vždy schopen správně odpovědět.
2. **Spolehlivost** – pokud uživatel tajnou informací nezná, není schopný ověřovatele přesvědčit o tom, že informaci zná.
3. **Nulová znalost** – pokud uživatel zná tajnou informaci, vždy přesvědčí ověřovatele o její znalosti bez toho, aniž by mu odhalil obsah tajné informace nebo jakékoli jiné informace. To znamená, že ověřovatel bude vědět jenom to, zda uživatel tajnou informaci zná nebo ne.

1.2 Sigma protokoly

Sigma protokoly ($\sum$ protokoly) jsou protokoly založené na stejném principu jako protokoly nulových znalostí, tzn. slouží k prokázání vlastnictví tajné informace pro přístup ke službě bez odhalení obsahu tajné informace. Oproti protokolům nulových znalostí jsou Sigma protokoly rychlejší a vhodnější pro praktické využití. Využívají se pro rozsáhle kryptografické aplikace, včetně schémat pro atributovou autentizaci, jsou implementovány ve spojitosti s kryptoměnami nebo se používají v protokolech elektronického hlasování.

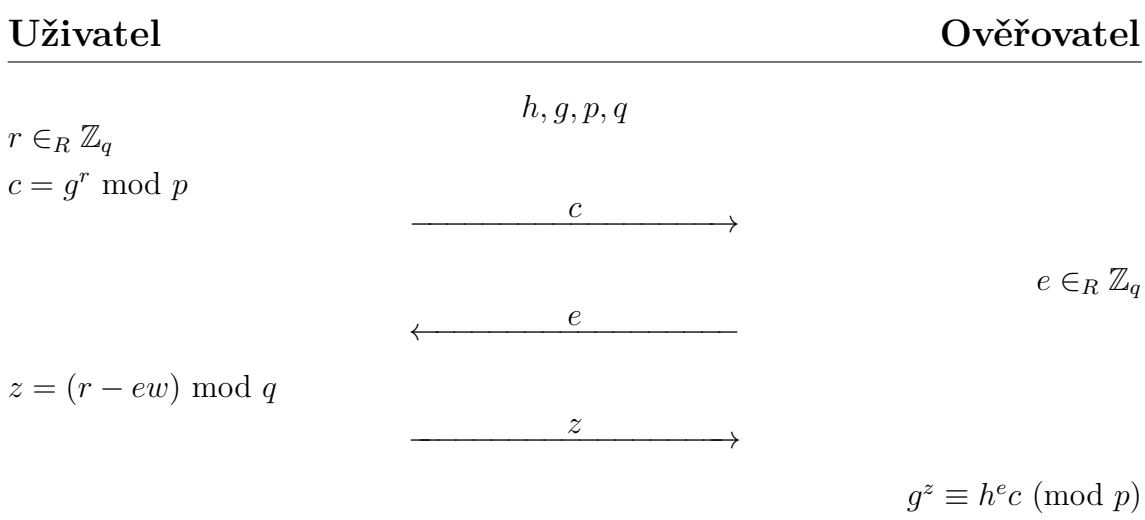
Základní vlastnosti Sigma protokolů vycházejí z vlastností protokolů s nulovými znalostmi, jsou však navíc doplněny o vlastnost třícestnosti. Protokol obecně obsahuje tři zprávy: závazek (*commit*), výzva (*challenge*), odpověď (*response*). Tyto zprávy jsou vyměňovány mezi uživatelem a ověřovatelem. V první fázi uživatel odešle *závazek*, v němž se zaváže k nějaké náhodné hodnotě. Ověřovatel tuto zprávu obdrží a uživateli zašle zprávu označovanou jako *výzva*. Uživatel na tuto zprávu, resp. výzvu, zareaguje a odešle *odpověď*. Obsahem této odpovědi je důkaz, že uživatel

opravdu zná tajnou informaci s respektováním zavázané hodnoty a výzvy. Uživatel na základě tohoto důkazu získá od ověřovatele přístup k požadované službě. Komunikace je zobrazena na obrázku 1.1.



Obr. 1.1: Třícestnost: Komunikace mezi uživatelem a ověřovatelem.

Schnorrův protokol [3] je jedním z nejznámějších a v praxi použitých zástupců Sigma protokolů (schéma je zobrazeno na obrázku 1.2). Hodnota $w \in \mathbb{Z}_q$ reprezentuje tajnou zprávu, hodnota h, kde $h = g^w$, je veřejný klíč uživatele, g je generátor, p prvočíselný je modulus a hodnota q vyjadřuje řád grupy. Uživatel se v prvním kroku zavazuje k náhodné hodnotě r a vypočítá veřejný závazek c. Obě tyto hodnoty posílá na stranu ověřovatele. V dalším kroku ověřovatel zašle uživateli výzvu e, na kterou očekává správnou odpověď. Uživatel vypočítá odpověď z, kde obsahem této odpovědi je i tajná zpráva w. V posledním kroku ověřovatel přijímá od uživatele odpověď z a ověřuje platnost $g^z \equiv h^e c \pmod{p}$. V případě, že by uživatel neznal tajnou zprávu w, obdržená odpověď by neodpovídala rovnici pro ověření validního uživatele. Schnorrův protokol je založen na problému diskrétního logaritmu a je použit např. v atributovém schématu U-Prove [1].



Obr. 1.2: Schéma Schnorrova protokolu.

1.3 Kryptografie nad eliptickými křivkami

Kryptografie eliptických křivek – Elliptic curve cryptography (ECC) je založena na algebraických strukturách nad konečným tělesem $\mathbb{F}_q$, kde $q = p^m$ označuje počet prvků, p je prvočíslo a m je přirozené číslo [4]. Řadí se mezi asymetrické kryptografické systémy a je postavena na principu matematického problému diskrétního logaritmu nad eliptickými křivkami. ECC se používá jako metoda pro šifrování veřejných klíčů.

ECC dosahuje stejné úrovně kryptografické bezpečnosti s použitím menší délky klíče než běžné kryptosystémy založené na problému diskrétního logaritmu v multiplikativní grupě [5]. Jsou také rychlejší, méně náročné na paměť a to je efektivní zejména pro méně výkonná zařízení, jako jsou například čipové karty.

Eliptická křivka je obecně definována rovnicí zvanou *Weierstrassův tvar*:

$$y^2 + 2xy = ax^3 + bx^2 + cx + d, \tag{1.1}$$

kterou lze dále zkrátit na tzv. *zkrácený Weierstrassův tvar*:

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

kde a a b jsou koeficienty určující tvar eliptické křivky, pro které platí $a, b \in \mathbb{F}_q$, x a y jsou souřadnice bodů vyhovující rovnici $y^2 \equiv x^3 + ax + b$.

Operace bilineárního párování

Operace bilineárního párování [6] na eliptické křivce je využívána u mnoha návrhů pokročilých asymetrických schémat. Funkcionalitou bilineárního párování, označovanou jako párovací funkce **e**, je mapování dvou prvků ze skupin prvočíselné aditivní cyklické grupy $\mathbb{G}_1$ a $\mathbb{G}_2$ do multiplikativní cyklické grupy $\mathbb{G}_T$.

Tato operace se řadí k výpočetně náročnějším a je zhruba 10krát pomalejší v porovnání s běžnými náročnějšími aritmetickými operacemi (např. mocnění nebo skalární součin bodu na eliptické křivce). Právě rychlost je důvod, proč se bilineární párování používá spíše u výpočetně výkonnějších zařízení.

1.4 Autentizace

Autentizace je proces, kdy je ověřována identita, případně oprávnění entity (uživatele). Cílem je rozhodnutí, zda bude mít daná entita umožněn přístup ke službě, datům nebo jiným informacím. Zajišťuje tak bezpečnost systému.

1.4.1 Metody autentizace

Autentizační techniky se dělí na tyto základní skupiny:

1. **Autentizace pomocí fyzického zařízení** – způsob, kdy uživatel ke svému prokázání používá fyzické zařízení, tzv. token. Může se jednat o *Universal Serial Bus* (USB) klíč, čipovou kartu apod. Tato zařízení jsou nositelem jedinečné tajné informace, například dostatečně silného a bezpečného hesla nebo kryptografickým klíčem. Některá zařízení jsou schopna provádět i kryptografické výpočty.
2. **Autentizace biometrikou** – do této skupiny spadají všechny formy autentizace, kdy se uživatel prokazuje svými biometrickými vlastnostmi – např. otisk prstu, sítnice oka, rozpoznávání obličeje, hlasu apod.
3. **Autentizace znalostí** – uživatel se prokazuje znalostí určité tajné informace, většinou hesla, které je sdíleno mezi uživatelem a ověřovatelem. Uživatel tuto informaci odhaluje v čitelné podobě a v případě neoprávněného odchycení je tak heslo snadno zneužitelné.
4. **Autentizace pomocí metody výzva-odpověď** – uživatel se prokazuje důkazem, že zná určité tajemství, jehož obsah neodhaluje. V průběhu komunikace ověřovatel zašle uživateli náhodnou výzvu, na kterou uživatel odpoví. Ověřovatel pak podle obsahu obdržené odpovědi rozhodne, zda je uživatel oprávněn k přístupu nebo ne.

5. **Autentizační metody s využitím kryptografie** – tyto metody vycházejí z technik důkazů nulových znalostí [7]. Příkladem jsou $\sum$ protokoly, které implementují metodu výzva-odpověď a neodhalují tak více informací.

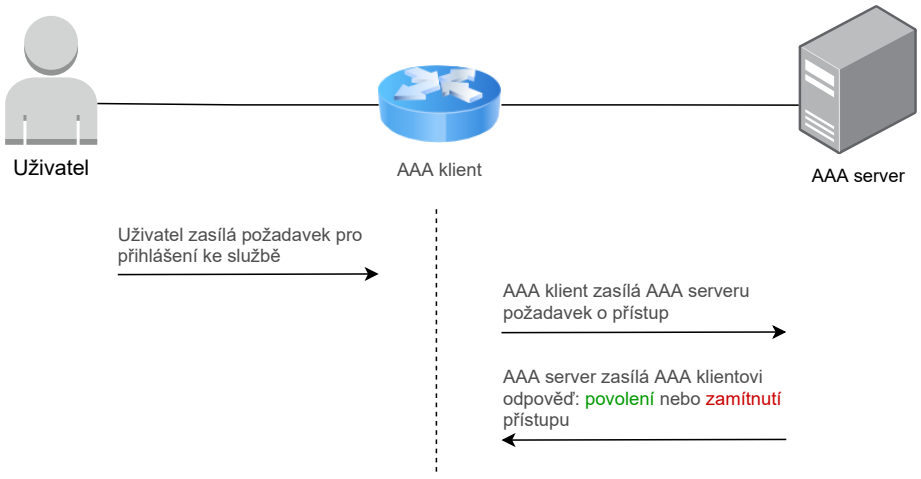
Další skupinou metod autentizace je atributová autentizace, kdy je identita uživatele anonymizována a zajištěna tak vyšší bezpečnost jeho osobních údajů a dat.

1.4.2 Autentizace versus autorizace

Pojem autentizace a autorizace bývá chybně zaměňován. Autorizace je proces, který následuje po autentizaci a kdy je entitě (uživateli) udělen přístup k požadované službě. Tento přístup je udělován důvěryhodnou autoritou, většinou ověřovatelem. Autorizace může mít omezení, například na určitý časový úsek, fyzickou lokaci apod.

1.4.3 Model AAA

Model *Authentication Authorization and Accounting* (AAA) označuje skupinu protokolů, které zprostředkovávají síťovou komunikaci. Autentizace a autorizace již byly v této kapitole popsány. Protokol *Účtování* (anglicky *Accounting*) poskytuje prostředky pro měření a zaznamenávání využívaní zdrojů uživatelem. Zaznamenávaná data můžou být informace o identitě uživatele, typů služeb a jejich provozu apod. Nejznámějšími AAA protokoly jsou *Remote Authentication Dial In User Service* (RADIUS) nebo *Terminal Access Controller Access-Control System* (TACACS). Na obrázku 1.3 je zobrazeno schéma komunikace mezi uživatelem, službou – AAA klientem a serverem poskytující AAA služby.

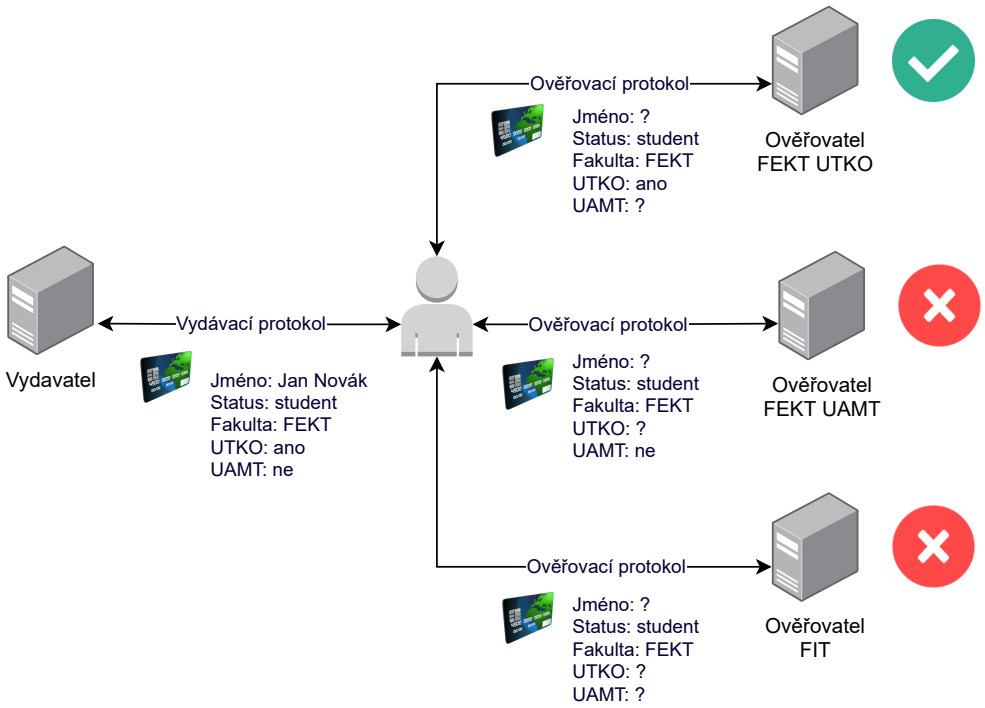


Obr. 1.3: Komunikace mezi uživatelem, AAA klientem a AAA serverem.

2 Atributová autentizace

Atributová autentizace je typ autentizace, která je založena na myšlence anonymizace uživatele a selektivního odhalování atributů. Atributová schémata využívají tzv. atributové pověření a jsou tak označována jako schémata využívající *Atributové pověření – Attribute-Based Credential* (ABC). Uživatel se při přístupu ke službě prokáže ověřovateli pouze patřičnými atributy, které jsou podepsané vydavatelem. Uživatel tak neodkrývá celou svou identitu a zároveň tak i ověřovatel uchovává pouze nezbytně nutná data.

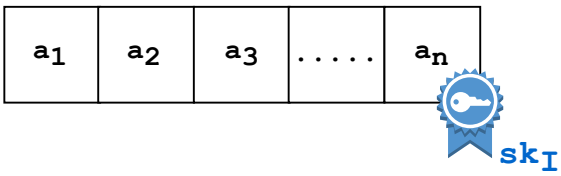
Praktické použití anonymních atributových schémat je relativně rozsáhlé. Typickým příkladem může být prokázání věku, prokázání se jako držitel platného oprávnění pro určité služby jako je sleva na jízdném nebo například přístup studenta k jednotlivým vnitřním prostorům v rámci univerzity, jak je uvedeno na obrázku 2.1). Princip je založen na tom, že se uživatel nemusí prokazovat již dalšími, pro ověřovatele nepotřebnými, údaji.



Obr. 2.1: Příklad atributové autentizace - vydávací a ověřovací protokol.

Atributové pověření

Atributové pověření [8] je množina atributů $(a_1, a_2, \ldots a_n)$, která je podepsaná důvěryhodnou autoritou – vydavatelem. Tímto atributovým pověřením pak uživatel prokazuje svůj oprávněný přístup ke službě. Pomocí selektivního odhalování atributů se z této množiny vůči ověřovateli prokazuje jen konkrétními atributy, které jsou ověřovatelem vyžádány (obrázek 2.2).



Obr. 2.2: Struktura atributového pověření.

2.1 Vlastnosti

Požadované vlastnosti schémat pro anonymní atributovou autentizaci jsou následující:

1. **Anonymita** – identita uživatele je v průběhu autentizace skryta.
2. **Nesledovatelnost** – důvěryhodná autorita, která vydává digitální pověření (vydavatel) není schopna vysledovat vlastníky atributů.
3. **Nespojitelnost relací** – autentizační relace uživatele jsou navzájem nespojitelné. Není možné uživatele na základě autentizačních relací profilovat a sledovat.
4. **Selektivní odhalování atributů** – uživatel předkládá ověřovateli množinu vybraných atributů, které jsou pro přístup požadovány.
5. **Nepřenositelnost** – uživatel není schopen svá pověření předat jinému uživateli, který by tak mohl získat neoprávněný přístup ke službě.
6. **Revokace** – pověření, která nejsou validní jsou revokační autoritou odebrána ze systému.
7. **Identifikace** – efektivní odhalení útočníků nebo jinak škodlivých uživatelů revokační autoritou.

2.2 Entity

Anonymní schémata, která využívají atributové pověření, obecně obsahují tyto entity:

1. **Vydavatel (V)** – důvěryhodná autorita, která vydává uživateli pověření (podepsané atributy). V případě přítomnosti revokační autority v atributovém systému spolupracují s cílem odhalení a blokování uživatelů, kteří nejsou oprávněni mít ke službě přístup.
2. **Uživatel (U)** – vlastní kryptografický token (např. čipovou kartu). Prokazuje se ověřovateli pověřením za účelem přístupu ke službě.
3. **Ověřovatel (O)** – ověřuje, zda je uživatel vlastníkem platných pověření.
4. **Revokační autorita (RA)** – důvěryhodná autorita, je přítomna u některých atributových systémů (RKVAC), nastavuje systémové parametry, spolupracuje s vydavatelem a ověřovatelem. Revokační autorita může rozhodovat o třech typech revokací:
   - **Revokace autentizačních tokenů** – token uživatele je odebrán ze systému, jeho identita však zůstává stále anonymní.
   - **Revokace nespojitelnosti relací** – jsou odhaleny všechny přístupy uživatele ke službě z minulosti, které můžou být analyzovány a na jejich základě odepřen další přístup. Identita uživatele není odhalena a je nadále anonymní.
   - **Revokace anonymity** – identita uživatele je zcela odhalena.

2.3 Popis obecných protokolů

Kromě entit atributová schémata pro svou funkčnost obsahují obecně tyto protokoly:

1. **Nastavení (Setup)** – generování parametrů systému, nastavení soukromých a veřejných klíčů.
2. **Vydávací protokol** – protokol pro vydání pověření na základě obdržených atributů od uživatele.
3. **Ověřovací protokol** – protokol pro ověření prokazovaného pověření od uživatele.
4. **Revokační protokol** – protokol, který slouží pro identifikaci uživatele a zneplatnění jeho přístupu ke službě. Po provedení revokace uživatele je další jeho pokus o přístup zamítnut.

Protože se atributová schémata od sebe navzájem liší, tak i jejich jednotlivé protokoly a jejich algoritmy můžou být od obecného popisu odlišné. Jedním z nejznámějších atributových schémat je U-Prove [1], který je vyvíjený firmou Microsoft. U-Prove schéma je založeno na problému diskrétního logaritmu a obsahuje vydávací

a ověřovací protokol. Schéma však nedisponuje funkcí nespojitelnosti relace a možnosti identifikace uživatele. Dalším zástupcem je schéma Identity Mixer [9]. Toto schéma je na rozdíl od U-Prove schopno akce nespojitelnosti relace a jeho návrh má v určité formě implementován revokační mechanismus. Ověření je platné po určitý časový úsek a v případě, že uživatel splňuje požadavky a nejeví se jako nebezpečný, je tento časový úsek prodloužen. Identity Mixer je založen na problému prvočíselné faktorizace. Stejně jako ve schématu U-Prove, tak i zde chybí identifikace uživatele a v praktickém využití je nevýhodou vyšší časová náročnost u prováděných operací.

2.4 Klíčově ověřitelná anonymní pověření

Klíčově ověřitelná anonymní pověření – Keyed-Verification Anonymous Credentials (KVAC) [10] je atributové schéma založené na principu algebraického *Message Authentication Code* (MAC) vycházejícího z podpisu *weak Boneh-Boyen* (wBB) a důkazech nulové znalosti. Návrh tohoto schématu je postaven na myšlence, kdy je vydavatel a ověřovatel jedna entita, to znamená, že tyto entity mezi sebou sdílejí soukromý klíč. Architektura tohoto schématu splňuje obecné požadavky atributové autentizace, tzn. schéma obsahuje entitu uživatele, který se prokazuje pověřením pro přístup ke službě, a dále pak entitu vydavatele a ověřovatele.

2.4.1 Protokoly

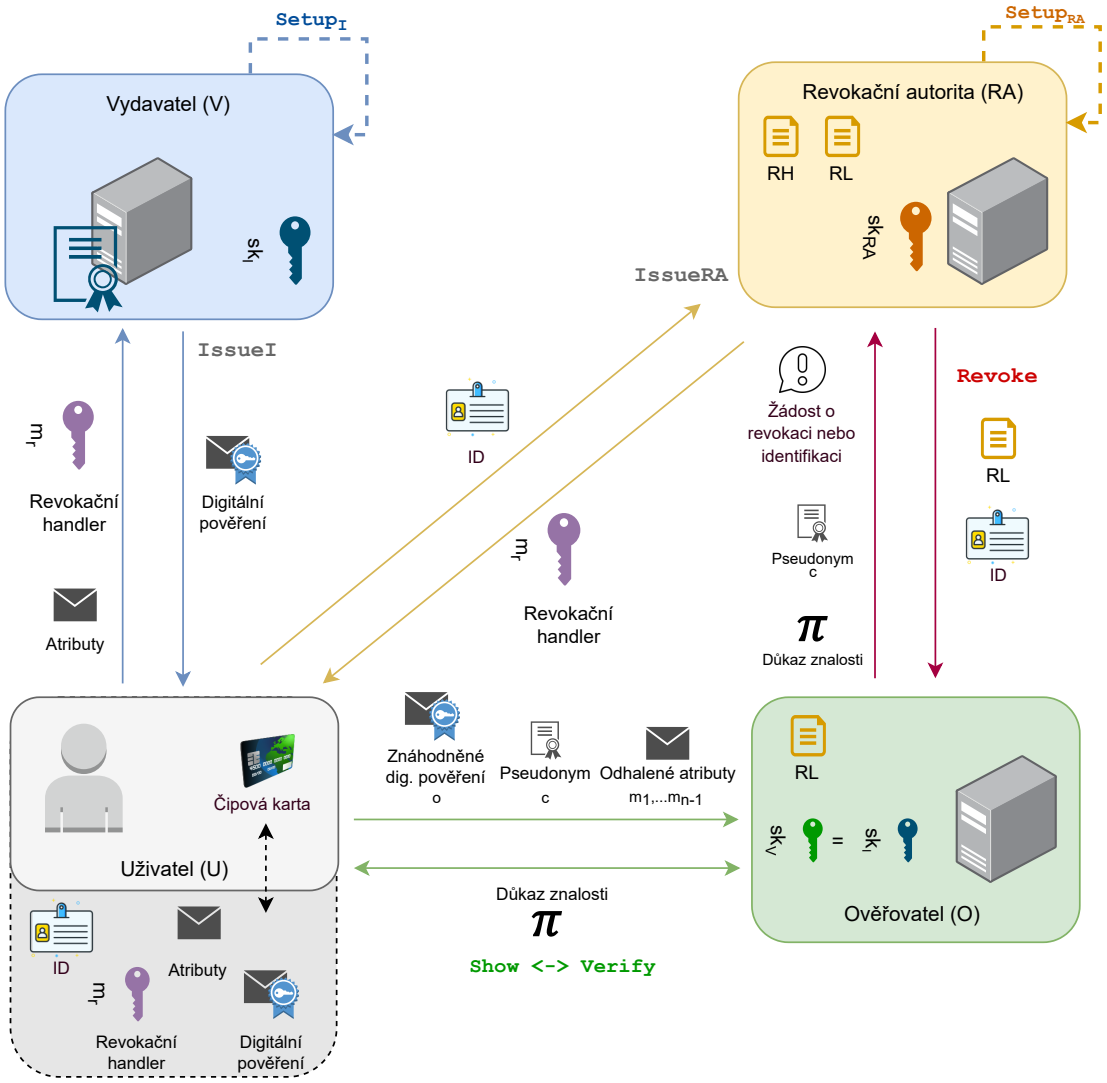
Schéma KVAC obsahuje základní protokoly pro schémata atributové autentizace, které jsou rozšířené o další vlastnosti a algoritmy:

1. **Nastavení (Setup)** – v této fázi jsou nastaveny systémové parametry $params_I$ a soukromé klíče vydavatele sk_I. Fáze nastavení obsahuje jeden algoritmus. Algoritmus `SetupI` má k dispozici vstupní bezpečnostní parametr κ. Je spuštěn na straně vydavatele a výstupem jsou vygenerované systémové parametry $params_I = (q, \mathbb{G}, g_1)$, kde $\mathbb{G}$ zastupuje cyklickou grupu, g_1 je bod generující eliptické křivky a hodnota q jeho řád, a množina soukromých klíčů $sk_I = (x_0, x_1, ..., x_n) \in_R \mathbb{Z}_q$.
2. **Vydávací protokol** – tento protokol obsahuje algoritmus `IssueI` a probíhá mezi uživatelem a vydavatelem. Uživatel zasílá své atributy $(m_1, ...m_n)$ vydavateli k podepsání a vydání atributového pověření. Vydavatel na základě vstupních parametrů $params_I$, sk_I a obdržených atributů vypočítá digitální atributové pověření $cred = (\sigma, \sigma_{x_0}, \sigma_{x_1}, ..., \sigma_{x_n})$, které zašle zpět uživateli. Po této fázi má již uživatel k dispozici pověření a může se jím prokázat ověřovací autoritě.

3. **Ověřovací protokol** – v rámci ověřovacího protokolu probíhá komunikace mezi uživatelem a ověřovatelem. Uživatel se ověřovateli prokazuje zvolenými odhalenými atributy. Tato fáze se skládá ze dvou algoritmů. Na vstup prvního algoritmu `Show` jsou dány parametry systému $params_I$, množina osobních atributů $(m_1, ...m_n)$ a jejich pověření *cred*. Ověřovatel uživateli zašle náhodnou hodnotu *nonce*. Z těchto hodnot uživatel sestaví důkaz znalosti $(\hat{\sigma}, \pi)$, který předkládá ověřovateli. V následujícím algoritmu `Verify` má ověřovatel k dispozici vstupní parametry $params_I$ a množinu klíčů vydavatele, kdy platí $sk_V = sk_I$. Ověřovatel ověří, zda je důkaz $\pi = (e, s_{m \notin D}, s_v)$, který získal od uživatele, validní. Pokud ano, uživatel je vyhodnocen jako entita, která má oprávnění pro přístup.

2.5 Klíčově ověřitelné anonymní pověření s revokací

Schéma atributové autentizace *Klíčově ověřitelná anonymní pověření s revokací – Revocable Keyed-Verification Anonymous Credentials* (RKVAC) vychází ze schématu KVAC [10]. Návrh tohoto schématu byl poskytnut vedoucím práce. Základní koncept je podobný s protokolem KVAC, oproti němu navíc disponuje entitou revokační autority a mechanismy pro efektivní revokaci, tzn. je možné ze systému odvolat podezřelého uživatele a pokud je potřeba, tak i odkrýt jeho identitu. RKVAC kromě entity revokační autority, stejně jako systém KVAC, obsahuje entitu uživatele, vydavatele a ověřovatele, kde soukromý klíč vydavatele a ověřovatele je shodný. Schéma RKVAC je zobrazeno na obrázku 2.3).



Obr. 2.3: Architektura systému RKVAC.

2.5.1 Protokoly

Princip protokolů vychází z protokolů schématu KVAC, které jsou navíc doplněny o další parametry a algoritmy.

Nastavení systému

Tato fáze obsahuje kromě algoritmu `SetupI` i algoritmus `SetupRA`, který je spouštěn revokační autoritou.

1. **SetupI** – vstupním parametrem je bezpečnostní parametr κ. Výstupem algoritmu je dvojice hodnot. První hodnotou jsou parametry systému $params_I$, kde $params_I = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$. V těchto parametrech je oproti sys-

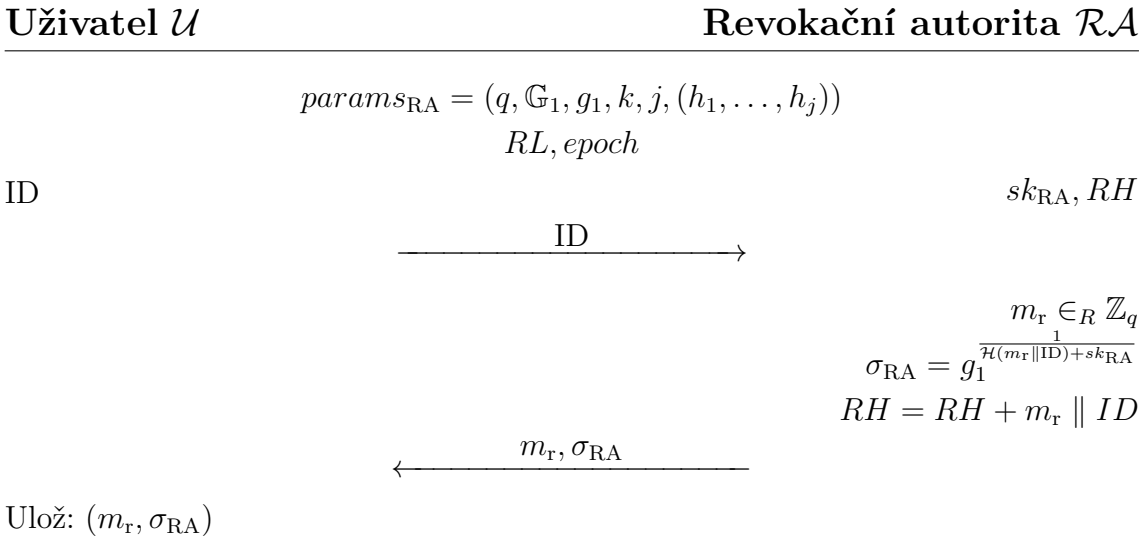
tému KVAC obsažen navíc parametr cyklické grupy $\mathbb{G}_2$ a $\mathbb{G}_T$, kde všechny cyklické grupy jsou stejného řádu generující bod eliptické křivky g_2 a také nedegenerativní mapu $\mathbf{e} : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$. Druhou hodnotou, která je výstupem tohoto algoritmu je množina klíčů vydavatele sk_I.

2. **SetupRA** – na vstupu algoritmu je bezpečnostní parametr κ a hodnota udávající maximální počet ověřovaných relací v určité časové epoše $ver_{max} = k^j$, kde celočíselná hodnota k určuje počet tzv. randomizérů e jejich pověření σ_e a celočíselná hodnota j označuje počet náhodných celočíselných proměnných α. Výstupem algoritmu jsou hodnoty $params_{RA}$, vygenerovaný pár klíčů sk_{RA} a pk_{RA} a prázdná Revokační Listina RL.

Vydávací protokol

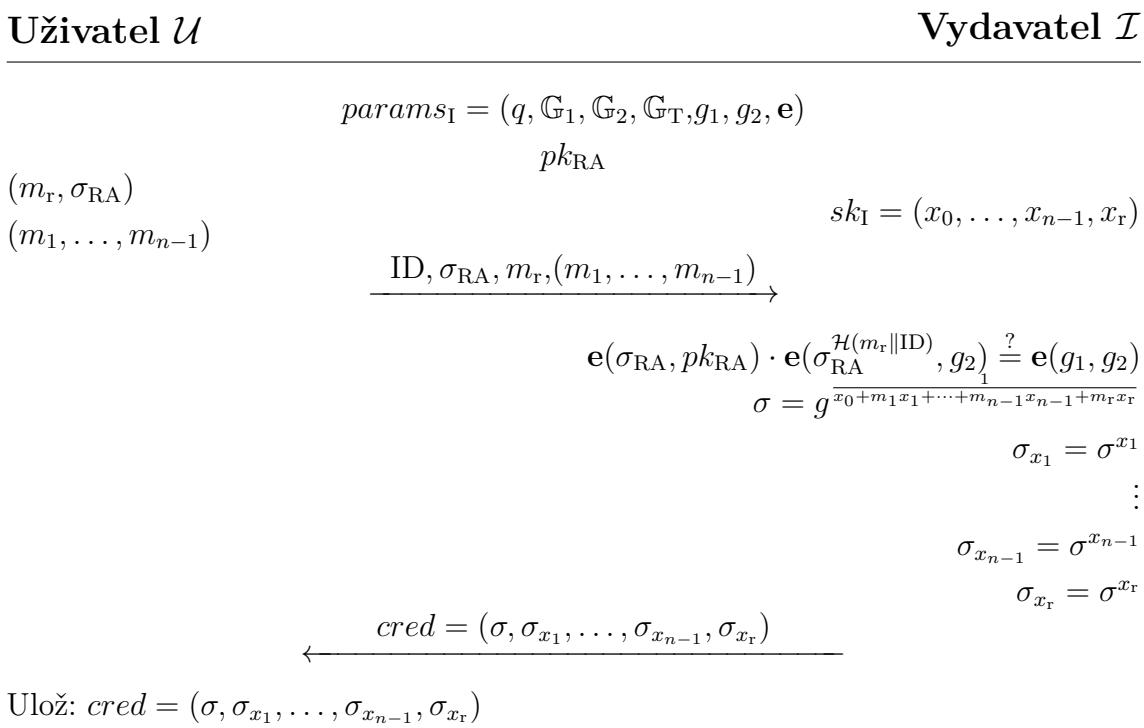
Vydávací protokol je rozdělen na dvě fáze. V první fázi probíhá komunikace mezi uživatelem a revokační autoritou – algoritmus `IssueRA`, v druhé fázi pak mezi uživatelem a vydavatelem – algoritmus `IssueI`.

1. **IssueRA** – v tomto algoritmu jsou vstupem systémové parametry revokační autority $params_{RA}$ a soukromý klíč sk_{RA}. Dalším vstupem je jednoznačný identifikátor uživatele ID, který je revokační autoritě zaslán. Revokační autorita tyto vstupy zpracuje a vygeneruje revokační handler m_r a jeho podpis σ_{RA}. Tyto dvě hodnoty jsou výstupem algoritmu a odesílány na stranu uživatele. Průběh komunikace je zobrazen na obrázku 2.4.



Obr. 2.4: Vydávací protokol – algoritmus `IssueRA` probíhající mezi uživatelem a revokační autoritou.

2. **IssueI** – algoritmus zajišťuje komunikaci mezi uživatelem a vydavatelem s cílem vydání pověření. Vstupem algoritmu je soukromý klíč vydavatele $sk_I = (x_0, \ldots, x_{n-1}, x_r)$, veřejný klíč revokační autority pk_{RA} a systémové parametry $params_I$. Uživatel zasílá vydavateli svůj identifikátor ID, své atributy $(m_1, \ldots, m_{n-1})$, které chce podepsat a dále také dříve získaný revokační handler m_r s jeho podpisem σ_{RA}. Vydavatel ověří platnost revokačního handleru a vypočítá digitální pověření $cred = (\sigma, \sigma_{x_0}, \sigma_{x_1}, \ldots, \sigma_{x_{n-1}})$, které zašle zpět uživateli. Komunikace je zobrazena na obrázku 2.5.



Obr. 2.5: Vydávací protokol – algoritmus `IssueI` zajišťující komunikaci a přenos dat mezi uživatelem a vydavatelem.

Ověřovací protokol

Funkcí tohoto protokolu je ověření, zda vydané pověření, kterým se uživatel prokazuje ověřovali, je platné a rozhodnutí, zda uživatel má ke službě přístup či nikoliv. Stejně jako u schématu KVAC, je i v tomto rozšířeném schématu ověřovací protokol rozdělen na dva algoritmy – `Show` a `Verify`. První algoritmus zajišťuje komunikaci mezi uživatelem a ověřovatelem, druhý pak pracuje na straně ověřovatele a rozhoduje o tom, zda je uživatel a jeho pověření validní. Komunikace obou algoritmů je popsána na obrázku 2.6.

1. **Show** – vstupem jsou systémové parametry $params_I$, $params_{RA}$, množina atributů uživatele $(m_1, \ldots, m_{n-1})$, revokační handler m_r a pověření $cred$. Ově-

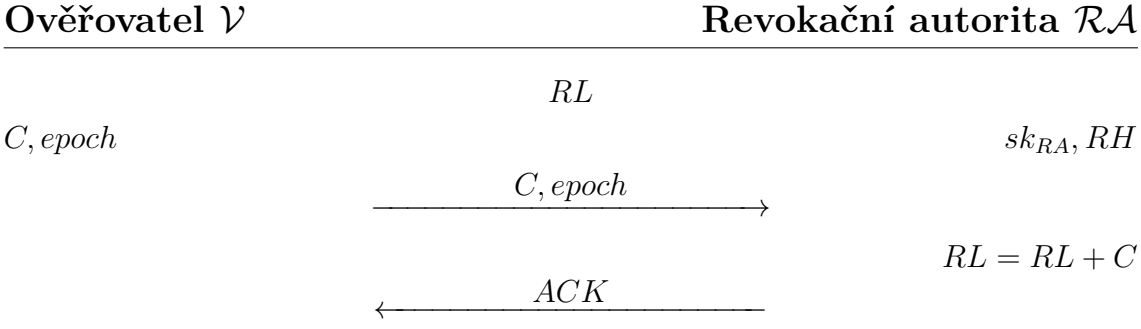
řovatel uživateli odešle náhodnou hodnotu *nonce* a identifikátor epochy *epoch*. Po obdržení těchto hodnot je sestrojen pseudonym C, vygenerované pomocné hodnoty pro vytvoření znáhodněného digitálního pověření $\hat{\sigma}$, důkaz znalosti π a důkaz znalosti konstrukce pseudonymu. Všechny tyto hodnoty, tj. pseudonym, znáhodněné digitální pověření, důkaz znalosti a atributy $m_{z \in D}$, které jsou odhaleny, uživatel posílá na stranu ověřovatele.

2. **Verify** – vstupem jsou hodnoty obdržené od uživatele z předchozího algoritmu, veřejný klíč revokační autority pk_{RA} a množina soukromých klíčů vydavatele sk_I, kdy platí, že tato množina klíčů je shodná s množinou soukromých klíčů ověřovatele ($sk_I = sk_V$). Ověřovatel na své straně zkontroluje, zda je důkaz znalosti skrytých atributů a důkaz znalosti konstrukce pseudonymu platný a jestli uživatel již není obsažen v seznamu uživatelů, kteří byli dříve revokovány revokační autoritou. Výstupem je tedy hodnota 0 nebo 1, podle toho, zda byl uživatel zamítnut nebo přijat.

Revokační protokol

Revokační protokol probíhá mezi ověřovatelem a revokační autoritou. Cílem je zneplatnit dříve vydaného pověření a zajistit, aby uživatel, který je podezřelý a vyhodnocen jako nevyhovující, neměl již nadále ke službě přístup. Zároveň tento protokol poskytuje i možnost odhalení identity uživatele.

Vstupem algoritmu je seznam revokačních handlerů RH, revokační listina RL, soukromý klíč revokační autority sk_{RA} a data přijatá ze strany ověřovatele, tj. žádost o revokaci nebo identifikaci uživatele, pseudonym C a důkaz znalosti π. Výstupem algoritmu je aktualizovaná revokační listina RL, která je zaslána zpět na stranu ověřovatele. Průběh revokačního protokolu je zobrazen na obrázku 2.7.



Obr. 2.7: Revokační protokol – schéma znázorňující komunikaci mezi ověřovatelem a revokační autoritou.

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

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

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

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

$\xleftarrow{nonce, epoch}$

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

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

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

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

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

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

Obr. 2.6: Ověřovací protokol obsahující algoritmus `Show` a `Verify`.

3 Nástroje pro vývoj aplikace

Tvorba a vývoj webových aplikací zahrnuje spoustu technologií, které jsou mezi sebou provázány. V praktické části práce bylo pro server a webovou aplikaci zvoleno a použito několik knihoven a balíčků tak, aby kód aplikace byl přehledný a její chod byl pro uživatele co nejvíce plynulý.

3.1 Aplikace a knihovny na straně serveru

Serverová část aplikace běží odděleně od webové aplikace. Úlohou je zpracovávání dat, se kterými dál pracuje webová aplikace. Na straně serveru jsou nainstalovány následující aplikace a balíčky.

3.1.1 Aplikace RKVAC

Tato aplikace provádí veškeré kryptografické výpočty a operace a zajišťuje nastavení systému, přidávání nových uživatelů, vydávání pověření a následně jejich ověření a revokaci podezřelých uživatelů. Aplikace byla poskytnuta vedoucím práce, je napsaná v jazyce C a je samostatně spustitelná přes rozhraní terminálu (obrázek 3.1). Pokud je při sestavení (tzv. build) aplikace pro spuštění nastaven parametr `-DRKVAC_PROTOCOL_REMOTE` na hodnotu `ON`, aplikace při posílání a přijímání *Application Protocol Data Unit* (APDU) příkazů a odpovědí očekává připojení vzdálené aplikace přes tzv. *Transmission Control Protocol* (TCP) socket. Ve výchozím nastavení je port socketu nastaven na 5000.

```
 [i] Card personalization started.
[+] Checking the user_list.csv ... file exists, opening user_list.csv
[+] Reading the last identifier from the user list.
[+] Checking if the increment identifier doesn't exist.

[+] Loading the user identifier to the remote card.
[i] Socket successfully created.
[i] Socket successfully binded.
[i] Server listening...
[i] Server accept the client 127.0.0.1:36356
[-] Command: APDU_SCARD_SELECT_APPLICATION
[+] Remote SCard request :
00 A4 04 00 07 76 75 74 32 31 30 31 00
[+] Remote SCard response:
90 00

[-] Command: INS_SET_USER_IDENTIFIER
[+] Remote SCard request :
70 0B 00 00 08 10 00 00 00 00 00 00 04
[+] Remote SCard response:
90 00

[+] Loading successful.
[+] Writing to the user_list.csv ... User Jan Novák with id 1000000000000004
was successfully added to the user list.
[i] Card personalization successful.
```

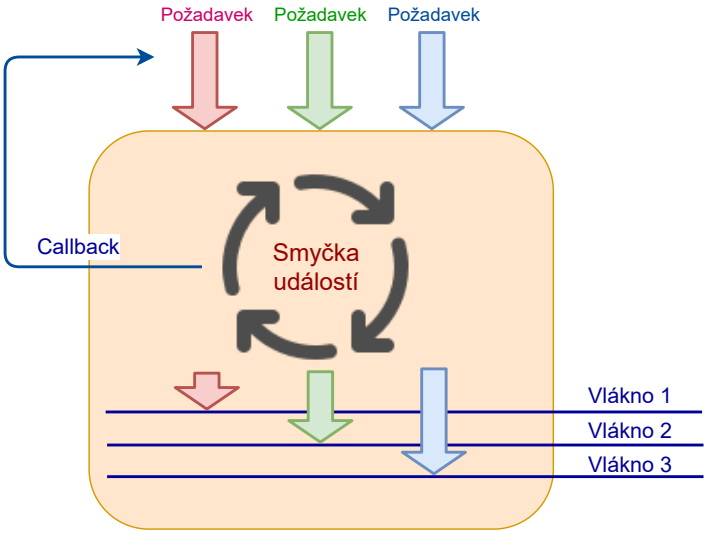
Obr. 3.1: Výpis aplikace RKVAC - vytvoření nového uživatele.

3.1.2 WebSocket

WebSocket je komunikační protokol, který přes spojení TCP zajišťuje obousměrnou komunikaci mezi dvěma uzly [11]. Tyto dva uzly jsou definovány *Internet Protocol* (IP) adresou a portem a přenos dat probíhá po celou dobu, kdy je spojení otevřené. WebSocket je navržen pro vytváření komunikačního kanálu mezi webovým prohlížečem a webovým serverem a v dnešní době je podporován hlavními prohlížeči jako je Chrome, Firefox, Safari, Opera apod.

3.1.3 Node.js

Node.js [12] je široce rozšířený open-source nástroj pro vytváření a správu dat na webovém serveru. Jeho využití je rozsáhlé. Node.js umí například dynamicky generovat soubory a pracovat s nimi, modifikovat data v databázi podle potřeby, případně shromažďovat data formulářů a další. Node.js je určen pro jazyk JavaScript a jeho funkcionalita je založena na tzv. smyčce událostí (angl. event loop). Do této smyčky přichází všechny požadavky z webové aplikace, které jsou posílány jednotlivě dál asynchronně spouštěným vláknům a poté zpracovávány. Pomocí tzv. callbacků se poté zpracovaný výstup vrací zpět. Obrázek 3.2 zobrazuje funkcionalitu NodeJS.



Obr. 3.2: Funkcionalita nástroje Node.js.

Součástí nástroje Node.js je balíčkovací nástroj npm [13]. Je to správce balíčků pro programovací jazyk JavaScript, pomocí něhož je možná instalace, správa nebo i distribuce JavaScriptových balíčků. Ve vývoji bývá již nepostradatelnou součástí a i v rámci této práce je často využíván.

Express.js

Express.js [14] webový aplikační framework pro Node.js, který umožňuje vytvářet rozhraní *Application Programming Interface* (API) mezi serverem psaným v Node.js a tzv. frontendovou částí webové aplikace. Je relativně jednoduchý pro použití a poskytuje mnoho funkcí, včetně všech *Hypertext Transfer Protocol* (HTTP) metod pro API komunikaci jako je např. `POST`, `GET`, `PUT` a další.

MomentJS a csv-parser

Tyto dva balíčky byly použity pro manipulaci s daty, které generuje RKVAC aplikace. Balíček MomentJS [15] obsahuje funkce pro formátování data a času, balíček csv-parser [16] umí přeformátovat data z formátu `csv` (případně s příponou `.log`) do formátu `JSON`, který je přívětivější z hlediska manipulace s daty. Tento balíček byl v aplikaci použit pro přeformátování dat ze souborů `user_list.csv`, `ra_rm_revoked.csv` a `ra_rh.csv` a `ve_requests.log`.

3.2 Knihovny pro webovou aplikaci

Webová aplikace je část aplikace, ke které uživatel přistupuje ze svého zařízení. V této části aplikace jsou proto implementovány balíčky sloužící k vytváření uživatelského rozhraní a dále je pak v rámci knihovny ReactJS implementováno rozhraní pro komunikaci se serverem.

3.2.1 ReactJS

ReactJS [17] je open-source framework, který vychází z jazyka JavaScript a umožňuje tvorbu uživatelského rozhraní a komponent. Je vyvíjen zejména společnosti Facebook a rozšířenou komunitou. Oproti čistému JavaScriptu má ReactJS přehlednější kód, nabízí jednoduchý zápis tříd a funkcí, ovládání stavů proměnných nebo komponent a je vhodný pro tvorbu tzv. „single-page“ aplikací.

Pomocí ReactJS lze jednotlivé vizuální prvky vytvářet jako tzv. komponenty, které se importují do větších celků. Výhodou je znovupoužitelnost, komponenty můžou být použity nesčetněkrát a pokud je potřeba udělat drobné úpravy a změny, vývojář upraví jen kód dané komponenty a změna se pak projeví všude, kde je komponenta použita.

Namísto běžného *Hypertext Markup Language* (HTML) syntaxu se pro psaní komponent využívá tzv. syntax *JavaScript XML* (JSX), který na první pohled připomíná HTML/XML syntax. Pro vývojáře je tento typ zápisu přehlednější než běžný

JavaScriptový zápis v kombinaci s HTML tagy. Prohlížeče však takový zápis neznají a proto se před spuštěním webové aplikace tento zápis kompiluje do čistého JavaScriptu.

styled-components

Je mnoho způsobů a knihoven, kterými lze nastavit *Cascading Style Sheets* (CSS) styly jednotlivých komponent. Jednou z takových knihoven je styled-components [18] – balíček pro JavaScript a jeho knihovny a frameworky. Na rozdíl od běžného nastavení CSS vlastností, které se mezi sebou většinou rozlišují podle selektorů (id, název třídy, apod.), u styled-components vývojář nastavuje vlastnosti stylů pro konkrétní komponentu, která je pak dále používána. Lze takto předejít případné nepřehlednosti v kódu.

GreenSock Animation Platform

Knihovna *GreenSock Animation Platform* (GSAP) je velice rozšířená, lze pomocí ní tvořit animace pro jednotlivé prvky a komponenty ve webové aplikaci [19]. Oproti běžným funkcím pro animace v jazyce JavaScript nabízí rozšířené možnosti animací a funkcí s nimi souvisejících. V případě projektů s náročnějšími a složitějšími animacemi je knihovna GSAP mnohonásobně výkonnější než běžné funkce pro animace v JavaScriptu.

i18n

Knihovna i18n [20] zajišťuje překlady aplikace. Je schopna automaticky lokalizovat uživatele, výchozí jazyk zařízení z nějž k aplikaci přistupuje a podle toho pro uživatele nejvhodněji zvolit výchozí jazyk.

Prettier a ESLint

Tyto dva balíčky nejsou pro plynulý chod aplikace stěžejní, nicméně pomáhají udržovat kód přehledný. Balíček Prettier [21] se stará o formátování kódu, tj. dodržení odsazení, počet znaků na jednom řádku, zajištění konvence jednoduchých a dvojitých uvozovek apod. Balíček ESLint [22] vývojáře upozorní na chyby v kódu, např. deklarování proměnné, která však není použita, chybějící středníky a další chyby. U obou těchto balíčků se konfigurují pravidla podle vlastních požadavků vývojáře a značně usnadňují práci s manuálním formátováním a kontrolou kódu.

3.3 Komunikace mezi webovou aplikací a kartou

Komunikace mezi webovou aplikací, prohlížečem a hardwarovým zařízením je poměrně komplikovaná na implementaci. Během dlouhodobého vývoje prohlížečů bylo vyvinuto několik možných variant, jak tuto komunikaci umožnit a to například následující:

1. **Netscape Plugin Application Programming Interface (NPAPI)** – toto aplikační rozhraní je v dnešní době velmi zastaralé a během let se jeho podpora postupně snižovala. Prohlížeč Google Chrome tuto platformu přestal podporovat od verze 45 (září 2015), Mozilla Firefox podporuje pouze Adobe Flash běžící na tomto rozhraní, nicméně i tato podpora bude brzy ukončena [23].
2. **Java applet** – chod Java appletu je závislý na rozhraní NPAPI, proto není možné sestavit komunikaci mezi stránkou a čipovou kartou prostřednictvím Java appletů.
3. **WebUSB** – moderní rozhraní, které slouží pro připojení zařízení k webové stránce. Z bezpečnostních důvodu však blokuje všechna zařízení, která mají implementováné *Chip Card Interface Device* (CCID) protokoly [27], proto je toto rozhraní pro tuto práci také nepoužitelné.
4. **Doplňky do prohlížeče** – použitelné řešení pro připojení čipových karet, využívají jej některé banky (např. Komerční banka) a jiné instituce. Většinou jsou to však proprietární řešení na klíč. Při průzkumu byly nalezeny dva volně dostupné doplňky:
   - **Smart Card Connector** – doplněk vyvíjený přímo firmou Google [28], použití je však omezeno na *Operační systém* (OS) ChromeOS. Pro použití na ostatních operačních systémech byl napsán rozšiřující kód [29], nicméně kód je zastaralý (poslední aktualizace v roce 2017) a jeho zprovoznění v rámci této práce nebylo úspěšné.
   - **Webcard** – open-source doplněk, který umožňuje nejen připojení čtečky, ale i APDU komunikaci. Instalace a nastavení doplňku má svá úskalí, pro použití ve webové aplikaci a v této práci je to však nejvhodnější řešení.

3.3.1 Webcard

Webcard [31] je open-source řešení a doplněk pro připojení čtečky k webové stránce v prohlížeči Google Chrome. S pomocí tohoto doplňku lze na kartu zasílat APDU příkazy a přijímat z ní APDU odpovědi. Použití doplňku je v omezeno na operační systémy MacOS a Windows.

Soubory doplňku jsou rozděleny do dvou podsložek:

1. **extension** – rozšíření, které je instalováno do prohlížeče,
2. **native** – zdrojové soubory pro kompilaci a instalační skript,

dále pak kořenová složka obsahující manuál na instalaci a spuštění doplňku a informace o licenci. Doplněk Webcard je pod licencí *Massachusetts Institute of Technology* (MIT).

Složka extension

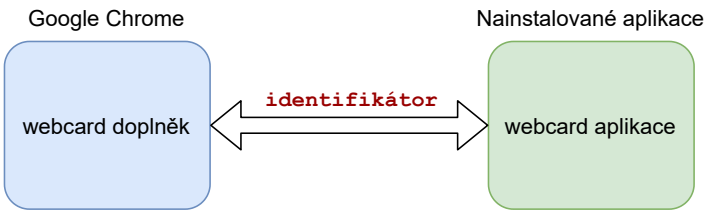
Vlastnosti a funkce doplňku jsou obecně definovány těmito soubory [32]:

1. `manifest.json` – obsahuje údaje o doplňku – např. název, popis, verze, stránky, na kterých je doplněk povolen apod.,
2. `background.js` – skript, který běží v prohlížeči na pozadí. Obsahuje funkci `chrome.runtime.onConnect.addListener()`, která běží při spuštěném doplňku a mimo jiné volá funkci `connectNative()`, která je taktéž obsažena v tomto souboru.
3. `content.js` – skript, který přistupuje k modelu *Document Object Model* (DOM) (objektově orientovaná reprezenace HTML dokumentu)[33] a k webové stránce.

Po instalaci doplňku do prohlížeče je vygenerován náhodný identifikátor, který je nutno definovat v souborech ve složce `native`. Podrobná instalace doplňku Webcard pro operační systém Windows je popsána v kapitole 4.4.3, pro operační systém MacOS pak v kapitole 4.5.2.

Složka native

Složka native obsahuje mimo zdrojové soubory pro kompilaci i další soubory, které jsou zahrnuty do procesu kompilace (`webcard.h`, `webcard.cpp`, `json.hpp`) a dále pak soubor `org.cardid.webcard.native`, který je výchozím souborem pro rozhraní Native messaging [34], které poskytuje propojení nainstalované aplikace s doplňkem v prohlížeči (schéma je zobrazeno na obrázku 3.3). V tomto souboru je definován klíč `allowed_origins`, jehož hodnota je právě identifikátor vygenerovaného nainstalovaného doplňku.

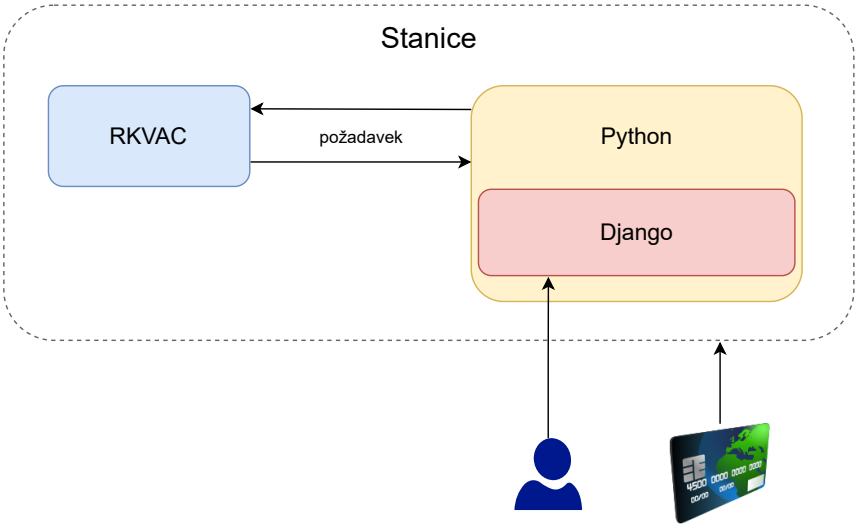


Obr. 3.3: Princip propojení Webcard doplňku a aplikace Webcard.

4 Implementace a spuštění aplikace

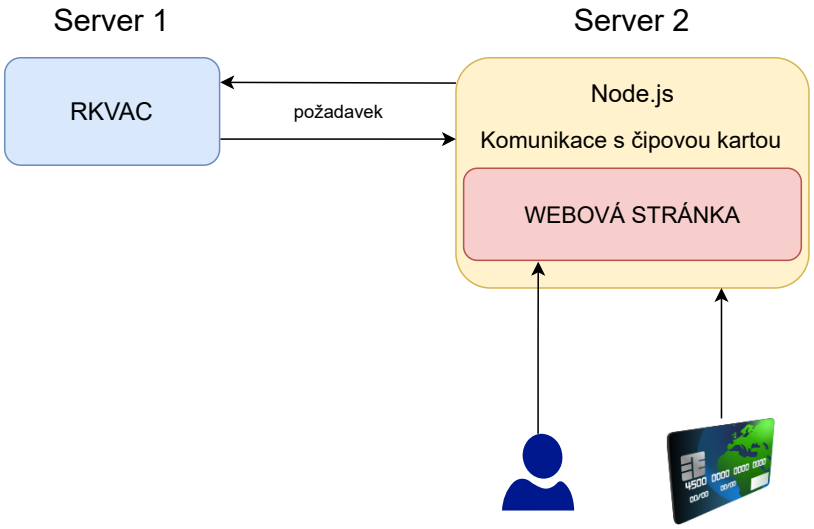
Součástí této práce je implementovaná aplikace, která vizualizuje fungování protokolu RKVAC a zároveň komunikuje s čipovou kartou. Během vypracování této praktické části bylo zamýšleno několik možných variant.

První zamýšlenou variantou bylo použití terminálové aplikace pro RKVAC v jazyce C v kombinaci s jazykem Python, jeho dostupnou knihovnou `subprocess` [24] a frameworkem Django. RKVAC aplikace je napsána v jazyce C a byla poskytnuta vedoucím práce. Tato varianta má však jeden hlavní nedostatek a to ten, že všechny tyto nástroje by musely být nainstalované a spuštěné na jedné stanici (obrázek 4.1). Tento návrh se však neshoduje s myšlenkou vytvoření a realizace aplikace, která by měla všechny výpočty pro RKVAC vykonávat na vzdáleném stroji a být jednoduše použitelná pro běžného uživatele z prohlížeče.



Obr. 4.1: Návrh implementace jako jedna stanice.

Druhou uvažovanou možností bylo rozdělení aplikace do dvou částí, tj. server a klient. Strana serveru by zahrnovala RKVAC aplikaci a, stejně jako v první variantě, funkce v jazyce Python s použitím knihovny `subprocess`. Výsledky by byly přes API rozhraní posílány na stranu klienta, na kterém by byl spuštěn druhý server, který by poskytoval i webové rozhraní pro uživatele. Pro toto řešení byla nalezena dostupná knihovna `node-pcsclite` [25], která obsahuje obalovací funkce pro rozhraní `pcsclite`, které komunikuje se čtečkou. V tomto případě by aplikace měla dvě serverové části. Po zhodnocení ani toto řešení nebylo vhodné. Návrh je zobrazen na obrázku 4.2.



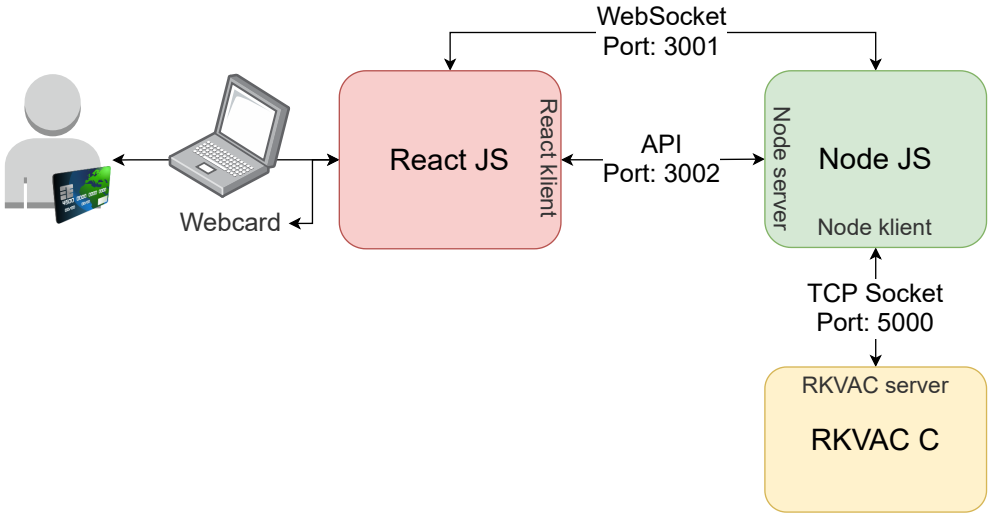
Obr. 4.2: Návrh implementace jako dvě serverové jednotky.

Třetím a nakonec i realizovaným návrhem byla možnost rozdělení celé aplikace na dva větší celky, kdy každý celek běží na vlastní stanici, tj. server a webová aplikace. Tyto dva celky mají mezi sebou dva komunikační kanály. Prvním je komunikace přes rozhraní API (ve výchozím nastavení port 3002), kterým si předávají data – požadavky (*requests*) a odpovědi (*responses*). Data, která jsou tímto rozhraním zasílána jsou např. údaje z formulářů vyplněné uživatelem na stránce nebo výpisy ze souborů, které se uživateli zobrazují na stránce. Druhým komunikačním kanálem je WebSocket spojení (port 3001), kdy část serveru v NodeJS funguje jako prostředník pro přenos APDU příkazů (*command*) a odpovědí (*response*) mezi klientem a RKVAC aplikací. APDU odpovědi, které NodeJS server na portu 3001 získá od uživatele, potažmo z čipové karty, zasílá dále na stranu RKVAC aplikace, která spouštěním jednotlivých příkazů (příkaz pro vytvoření nového uživatele, vydání pověření apod.) otevírá TCP spojení a naslouchá na portu 5000. Tato funkcionalita je funkční i v opačném pořadí, kdy RKVAC aplikace posílá APDU příkazy a NodeJS server je přeposílá přes WebSocket na stranu uživatele.

Webová aplikace poskytuje uživatelské rozhraní, vizualizaci systému RKVAC a komunikuje se čtečku čipových karet. Tato část je funkční s použitím doplňku Webcard popisovaným v kapitole 3.3.1 (a jeho související aplikací, která je nainstalována na stanici uživatele). Schéma architektury je zobrazeno na obrázku 4.3.

Toto řešení, stejně jako obě dvě předchozí, má své nevýhody a to především v doplňku Webcard (viz kapitola 3.3). Jeho instalace a konfigurace patří k náročnějším operacím (především na OS Windows), použití doplňku je omezeno pouze pro prohlížeč Google Chrome a samotná Webcard aplikace, která řídí přenos APDU dat, je

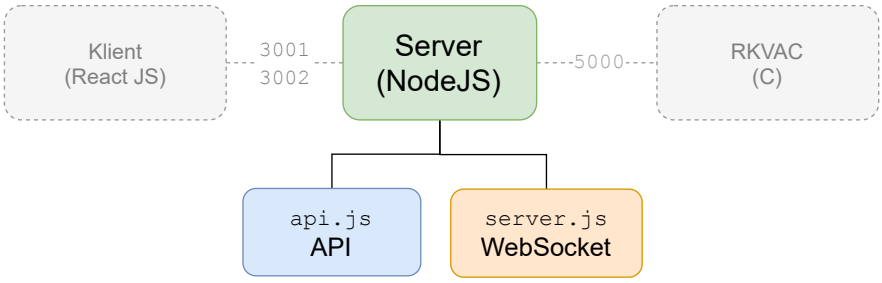
spustitelná pouze na OS Windows nebo MacOS. I přes tyto nedostatky je toto řešení nejoptimálnější a splňuje myšlenku, kdy karta komunikuje pouze s webovou aplikací. Uživatel tak kromě instalace doplňku a jeho aplikace (s přehlédnutím toho, že instalace Webcardu je komplikovanější) nemusí řešit instalaci žádných dalších aplikací. Samotná webová aplikace nemusí být umístěna přímo na počítači uživatele, může běžet na vzdálené stanici a uživatel k ní může přistupovat přes síť.



Obr. 4.3: Návrh architektury systému.

4.1 Server

Server zajišťuje poskytování APDU příkazů, které jsou zasílány na stranu webové aplikace (obrázek 4.4). Tato část je postavena na Node.js (viz kapitola 3.1.3) společně s (pro něj vyvíjeným frameworkem) Express.js a obsahuje dva hlavní soubory – `server.js` a `api.js`.



Obr. 4.4: Serverová část aplikace.

V souboru `server.js`, který je zároveň spustitelným souborem pro serverovou část, jsou obsaženy funkce pro WebSocket spojení. WebSocket na straně serverové části běží jako prostředník pro komunikaci mezi klientem a RKVAC aplikací.

Druhým souborem, který v této části figuruje, je `api.js`. Tento soubor přijímá API požadavky a data ze strany klienta, spouští RKVAC aplikaci s parametry, které od klienta získá a také zpět klientovi zasílá zpracovaná data. Funkce, které se v tomto souboru nachází lze rozdělit podle použití do dvou skupin:

1. **Pomocné funkce** – tyto funkce jsou volány v API funkcích (viz další bod). Většinou se jedná o funkce, které kontrolují, zda existuje určitá složka a jestli obsahuje nějaké soubory. Dále pak do této skupiny patří funkce, které vrací informace o epochách nacházejících se v systému.
2. **API funkce** – tyto funkce jsou spouštěny API voláním ze strany klienta přes příslušnou *cestu*. Buď se jedná o funkce pro jednotlivé protokoly (Nastavení a personalizace, Vydání, Ověření, Revokace), které spouští aplikace RKVAC, nebo funkce, které zpracovávají data z logů, které aplikace RKVAC během svého chodu generuje (např. `getVerifyResult`, `revokeList`, `epochList` a podobně). Speciálním typem funkce je `eraseDB`, která je také volána klientem a jejím úkolem je vymazání všech dat (vydána pověření, ověřovací log, údaje o epochách a rekovovaných uživatelích atd.) z RKVAC aplikace.

Funkce, které klient volá jsou API funkce a všechny mají podobnou strukturu. Tato práce obsahuje dva typy – `app.post` nebo `app.get`, což jsou funkce naimportovány z knihovny Express.js. Oba dva typy funkcí mají dva vstupní argumenty. Prvním je název *cesty* – pokud je na straně klienta volán požadavek s touto cestou, tak na straně serveru je volána příslušná funkce, která má zaregistrovanou tuto cestu. Druhým argumentem funkce je vnořená funkce se vstupním parametrem `req`, který zastupuje požadavek (*request*) a proměnnou `res`, která obsahuje data pro odpověď (*response*) a je zasílána zpět klientovi.

Vnořená funkce již obsahuje sled příkazů a případně dalších funkcí. Stěžejním je funkce `exec`, která s odpovídajícími parametry ve formátu textového řetězce slouží pro spouštění příkazů aplikace RKVAC.

Funkce `exec` má tři vstupní parametry, prvním je příkaz, který má být spuštěn, druhým je objekt obsahující konfigurační parametry – v případě této aplikace je to nastavení parametru `timeout`, který určuje jak dlouho bude spuštěný proces běžet, dokud nebude ukončen. Hodnota `timeout` je u každé `exec` funkce nastavena na 45 sekund. Třetím argumentem je vnořená *callback* funkce. Tato funkce obsahuje tři argumenty – `error`, `stderr`, `stdout`. Pokud během spuštěného příkazu nastane chyba, je její výpis přiřazen argumentu `error`. Další argumenty `stderr` a `stdout` odpovídají základní konvenci, tzn. standardní chybový výstup a standardní výstup. Každá tato funkce je ukončena zasláním klientovi nastavené odpovědi `res` z nad-

řazené funkce, která obsahuje číslo odpovědi (200 v případě úspěšného dokončení, 400 nebo 500 v případě chyby), v některých případech zasláním dat klientovi (např. výpis revokovaných uživatelů nebo v případě chyby odpověď obsahuje i informaci o tom, jaký typ chyby nastal). Pokud je funkce ukončena chybou, je serverová část v NodeJS automaticky restartována. Ukázka funkce pro vytvoření nového uživatele je zobrazena ve výpise 1.

```
app.post('/person', (req, res) => {
  const command = `../rkvac-protocol-multos-1.0.0 -p -n
  ↪ ${req.body.name} -s ${req.body.surname}`;
  exec(command, { timeout: execTimeout }, (error, stdout, stderr)
  ↪ => {
    if (error) {
      console.log('error: \n', error);
      if (error.message.includes(SOCKET_ERR)) {
        res.sendStatus(500);
        resetServer();
      }
    }
    if (stderr) {
      console.log('stderr: \n', stderr);
      return;
    }
    console.log('stdout: \n', stdout);
    res.sendStatus(200);
  });
});
```

Výpis 1: Ukázka funkce pro API volání v souboru `api.js` – vytvoření nového uživatele (personalizce).

Tato funkce provádí v aplikaci RKVAC personalizaci uživatele. Jak už typ volané funkce napovídá, jedná se o požadavek `POST` s cestou `/person`. V druhém argumentu, kterým je vnořená funkce, je v konstantě `command` definován příkaz, který bude funkcí `exec` volán. V tomto případě tento příkaz obsahuje jméno a příjmení, které uživatel napsal na stránce do formuláře. Následuje volání funkce `exec` a pokud v RKVAC aplikaci, která je spuštěna, nastane chyba a její výpis obsahuje textový řetězec, jenž je v souboru `api.js` definován v konstantě `SOCKER_ERR`, je na stranu klienta odeslána odpověď se statusem 500 a restartován NodeJS server.

Pokud chyba nenastane, klient obdrží od serveru odpověď se statusem 200. Celá funkce je doplněna o funkci `console.log()`, která výpis z aplikace RKVAC zobrazí v konzoli serveru.

4.2 Webová aplikace

Úkolem webové aplikace je zajištění komunikace mezi uživatelem a čtečkou čipových karet (čipovou kartou) a pro její kompletní běh je vyžadována instalace prohlížeče Google Chrome a doplňku Webcard (viz kapitola 3.3.1). Na pozadí pak mezi webovou aplikací a serverem probíhá komunikace přes API rozhraní – z pohledu klienta zasílání požadavků a přijímání odpovědí ze strany serveru a posílání APDU příkazů a odpovědí mezi klientem a čipovou kartou. API komunikace je zajištěna pomocí API rozhraní, které je definované skrz adresu a port a nakonfigurováné v souboru `package.json` (viz výpis 2). Ve výchozím nastavení je jako adresa nastaven `localhost` a výchozí port pro komunikaci je 3002. V případě potřeby je možné v souboru `package.json` tuto adresu a port změnit (pak je nutné číslo portu také změnit na straně serveru v souboru soubor `api.js`).

```
1 {
2     "proxy": "http://localhost:3002",
3 }
```

Výpis 2: Část souboru package.json na straně webové aplikace.

Funkce pro komunikaci s API jsou definované zvlášť v každém souboru pro příslušný protokol (soubor `DemoSetup.js`, `DemoIssue.js`, `DemoVerify.js`, `DemoRevoke.js`). Všechny tyto funkce jsou napsány s využitím asynchronního programování a tzv. *Promise*. Promise fungují jako „příslib“ hodnoty, kterou metoda *někdy* vrátí. Tento příslib je definován funkcí `fetch`. Za touto funkcí následuje blok `then`, který přebírá vrácenou hodnotu ze serveru. Ukázka kódu pro nastavení systému je zobrazena ve funkci `setupSystem` ve výpise 3.

```
const setupSystem = async (name, surname) => {
    setResult('loading');
    props.disabledTabs(true);
    run(props.selectedReader);
    await fetch('/person', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
      },
      body: JSON.stringify({
        name: name,
        surname: surname,
      }),
    }).then(async (response) => {
      const resStatus = response.status;
      props.disabledTabs(false);
      if (resStatus === 200) {
        setResult('ok');
        setWarningMessage(false);
        props.onChange(true);
      } else {
        setResult('nok');
        setWarningMessage(true);
        props.onChange(false);
      }
    });
  };
```

Výpis 3: Část souboru DemoSetup.js – volání API pro vytvoření nového uživatele.

V první části této funkce probíhá změna stavů – `setResult('loading')` a `props.disabledTabs(true)`. První popisovaný stav – `setResult('loading')` zajišťuje, že se uživateli zobrazí na stránce „načítací“ stavová ikona, druhý stav zajišťuje to, že všechny záložky protokolů na stránce přejdou do stavu deaktivace a nelze na ně klikat do doby, než je požadavek ve funkci `fetch` ukončen a stav opět změněn. Funkce `run(props.selectedReader)` navazuje spojení pro komunikaci se zvolenou čtečkou čipových karet, čipovou kartou a otevírá WebSocket spojení na straně klienta. Další část obsahuje funkci `fetch`, která obsahuje typ požadavku (`POST`), hla-

vičku a obsah dat posílaných na server přes API rozhraní. V tomto případě se jedná o jméno a příjmení, které uživatel vyplnil do formuláře. Blok `then` přijímá ze strany serveru odpověď, konkrétně `status`. Status je číselná hodnota – 200 v případě, že na straně serveru proběhl požadavek podle očekávání a bez chyb, nebo pokud nastala chyba, obdrží klient hodnotu 500. Dále následuje podmínka, kde vstupem je obdržená hodnota `status` a podle vyhodnocení je provedena změna stavové ikony (uživatel na stránce uvidí buď zelenou fajfku nebo červený křížek), případně jestli se uživateli zobrazí nad formulářem chybová hláška (`setWarningMessage`) a jestli se spustí animace (`props.onChange`). Animace se v případě chyby nespouští, přestože uživatel na stránce zaškrtne políčko pro animaci.

Uživatelské rozhraní aplikace je postaveno na frameworku ReactJS (viz kapitola 3.2.1), balíčcích pro ReactJS – zejména `styled-components` a `GSAP`. Jednotlivé prvky webu (tlačítka, karty, textové vstupy, apod.) jsou naprogramovány jako komponenty, ze kterých je následně poskládán větší celek. Každý takový prvek má přiřazen styly a vlastnosti.

Příkladem těch nejzákladnějších komponent, které vychází pouze z jednoho elementu, může být tlačítko umístěné v menu nabídce (`MenuButton.js`), dlaždice (`Tile.js`) nebo komponenta pro ikonu (`Icon.js`). Mezi komponenty, které se skládají z více elementů a menších komponent, se může řadit například dlaždice pro výběr čtečky čipových karet (`CardReader.js`), tlačítko (*button* – `Button.js`) nebo zaškrtávací pole (*checkbox* – `Checkbox.js`). Komponentami, které jsou relativně obsáhlejší a fungují jako větší celek, je komponenta (`Canvas.js`) pro animace nebo komponenty, které zastupují jednotlivé protokoly v kategorii Demo (soubor `DemoSetup.js`, `DemoIssue.js`, `DemoVerify.js`, `DemoRevoke.js`).

Dalšími „stavebními“ prvky webové aplikace jsou stránky pro jednotlivé kategorie (složka `/pages`), pomocné funkce a konstanty pro styly (složka `/helpers`) nebo soubory pro jazykové překlady aplikace (složka `/assets`).

Ve výpise 4 jsou zobrazeny definované styly pro komponentu `LoadingImg`, což je ikona, která zobrazuje stav načítání. První dva řádky jsou vyhrazené pro import knihoven, třetí řádek importuje soubor `loading.gif`, což je samotný obrázek ikony. Konstanta `Wrapper` obsahuje element `div`, který je obsahuje styly pro pozicování elementu (`display`, `justify-content`, `align-items` a `padding`) a dále styly pro všechny „potomky“ elementy `img`. V tomto případě všechny elementy `img`, které budou vnořené do tohoto divu, budou mít výšku (`height`) a šířku (`width`) o velikosti `3rem`.

```
import React from 'react';
import styled from 'styled-components';
import loading from '../images/loading.gif';

const Wrapper = styled.div`
  display: flex;
  justify-content: center;
  align-items: center;
  padding: 2rem 0;

  img {
    height: 3rem;
    width: 3rem;
  }
`;

const LoadingImg = () => (
  <Wrapper>
    <img src={loading} alt="loading" />
  </Wrapper>
);

export default LoadingImg;
```

Výpis 4: Kód komponenty pro ikonu znázorňující načítání.

Tento zápis stylů, který je přiřazen určité konstantě, je typickým zápisem pro použitou knihovnu `styled-components` a konstanta `Wrapper` je použitelná jako nastylovaná komponenta. Druhá konstanta `LoadingImg` je komponenta, která obsahuje nastylovaný div a obrázek načítající ikony. Poslední řádek definuje, že konstanta a komponenta `LoadingImg` je „exportována“ a může být naimportována v jiných souborech (narozdíl od komponenty `Wrapper`, kterou lze použít jen lokálně v rámci tohoto souboru).

Častokrát se CSS vlastnosti a styly v jednotlivých komponentách opakují. Aby byly tyto opakující se vlastnosti konzistentní i v případě, kdy je zapotřebí provést úpravu (např. změnit definici barvy), jsou definovány jako konstanty v souboru `themeConstants.js`. Změna se pak projeví všude, kde je daná konstanta použita. Část souboru je zobrazena ve výpise 5.

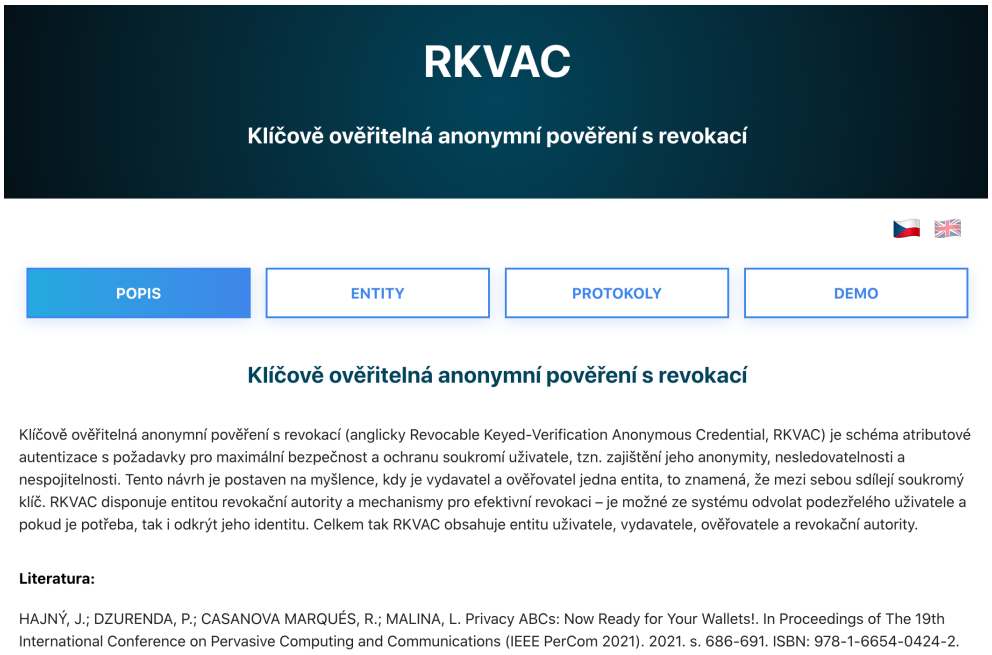
```
const colors = {
  dark: '#000',
  white: '#FFF',
  success: 'green',
  ...
  user: {
    bg: '#ffeaea',
    border: '#d46868',
  }
};
const boxShadow = {
  blueLight: '0 4px 15px 0 rgba(65, 132, 234, 0.2)',
  ...
};
const transition = 'all .4s ease-in-out';
const breakPoint = {
  xs: '27.5rem',
  ...
};

export default {
  colors,
  boxShadow,
  transition,
  breakPoint,
};
```

Výpis 5: Konstanty pro CSS vlastnosti – soubor `themeConstants.js`.

Soubor `themeConstants.js` má 4 hlavní konstanty – *colors* s definicemi barev, *boxShadow* s definicemi stínů implementovaných např. u tlačítek, *transition* pro plynulý přechod při změně CSS vlastnosti na stránce a *breakPoint*, tj. hodnoty pro rozlišení, při kterém nastává změna rozložení prvků na stránce (responzivita).

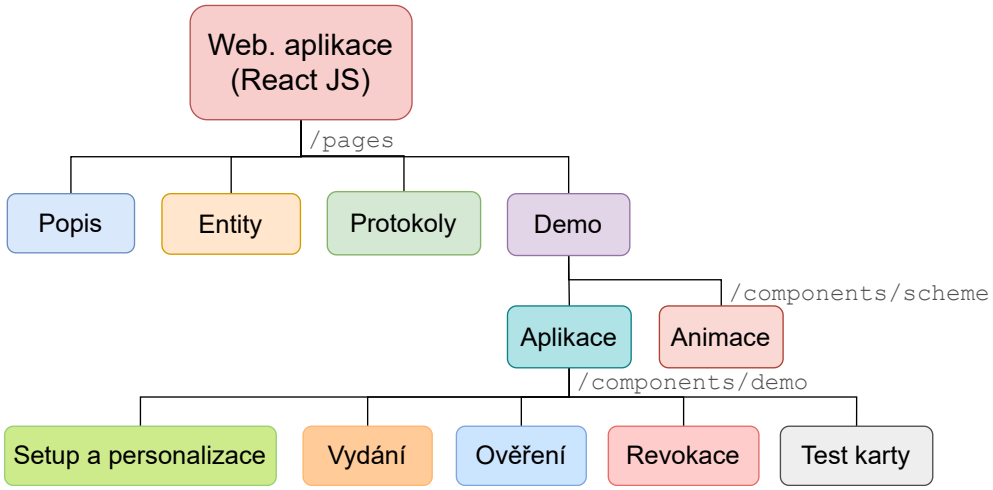
Vizuální část aplikace (obrázek 4.5), která je už přímo viditelná pro uživatele je rozdělena do čtyř základních kategorií:

1. **Popis** – obecný popis RKVAC protokolu,
2. **Entity** – popis všech entit a jejich úloh, které se v protokolu nachází,
3. **Protokoly** – popis jednotlivých protokolů,
4. **Demo** – demonstrátor a vizualizace průběhu RKVAC protokolu. Tato kategorie obsahuje další rozdělení podle jednotlivých protokolů a sekci pro otestování připojené čtečky a komunikace s kartou.



Obr. 4.5: Webová aplikace.

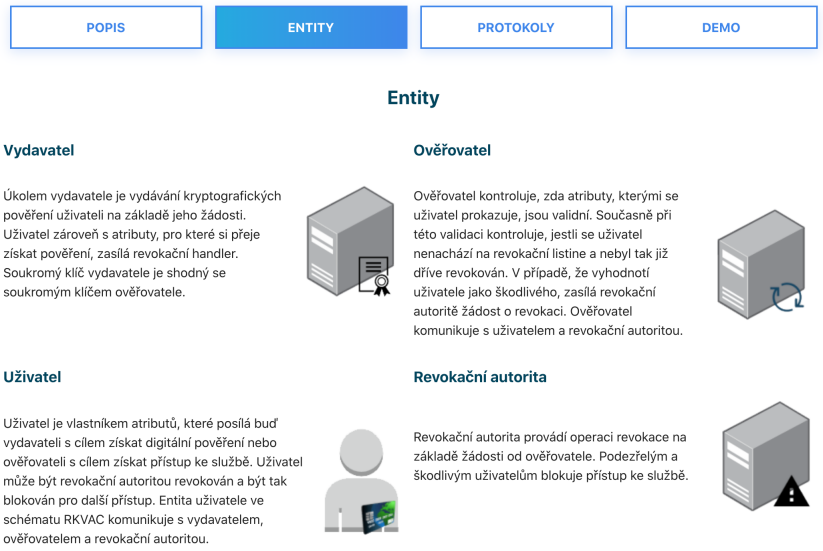
Rozdělení těchto kategorií a obsahu webu je zobrazeno ve stromové struktuře na obrázku 4.6. Celá webová aplikace má dva překlady, jeden v češtině a druhý v angličtině. Jazyk lze přepínat kliknutím na vlajky, které reprezentují daný jazyk a překlad a nachází se nad kategoriemi. V případě, že operační systém, ve kterém si uživatel aplikaci prohlíží, nepodporuje zobrazení těchto vlajek ve výchozím nastavení (např. OS Windows), jsou tyto vlajky alternativně nahrazeny textem – `CZ` a `EN`.



Obr. 4.6: Stromová struktura webové aplikace.

Kategorie; Popis, Entity a Protokoly

V kategorii Popis si uživatel může přečíst základní informace o systému RKVAC, jeho výhody a vlastnosti. Pro lepší pochopení a orientaci je zobrazeno i statické schéma. Následující dvě kategorie, Entity (obrázek 4.7) a Protokoly, obsahují obecný popis všech entit, které se v systému vyskytují a popis jednotlivých protokolů, včetně jejich schémat.

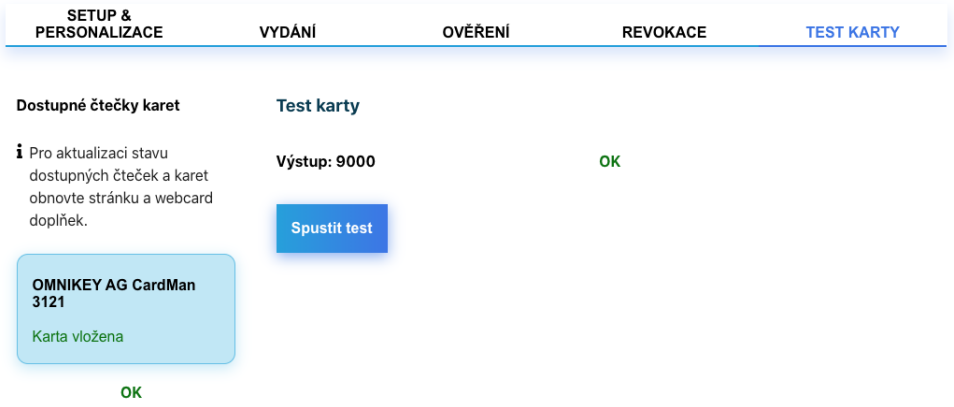


Obr. 4.7: Kategorie Entity.

Kategorie; Demo

Kategorie Demo je vizuálně rozdělena na dvě části. První část je vyhrazena pro komunikaci s aplikací RKVAC v jazyce C a čipovou kartou. Tato část slouží jako grafické uživatelské rozhraní mezi uživatelem a RKVAC aplikací. Druhá část obsahuje prostor pro animovanou vizualizaci.

V první popisované části jsou záložky *Setup a Personalizace, Vydání, Ověření* a *Revokace*, které zastupují jednotlivé protokoly systému RKVAC a poslední záložka, *Test Karty*, umožňuje uživateli si ověřit, zda je čtečka čipových karet a čipová karta připojena a zda probíhá komunikace (obrázek 4.8).

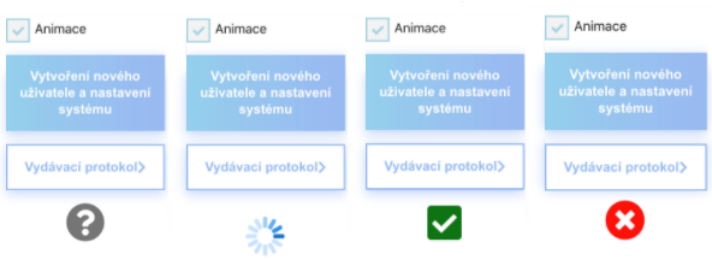


Obr. 4.8: Záložka test karty.

Pod záložkami se v levé části nachází výpis připojených čteček čipových karet, kde u každé položky (čtečky) je doplňující informace, zda je v ní vložena karta nebo ne. Protože doplněk Webcard není schopen detekovat v reálném čase odpojení a připojení čtečky nebo vložení a vyjmutí karty, je uživatel nad seznamem čteček informován o tom, že pro aktualizaci stavu čteček musí obnovit prohlížeč a doplněk Webcard.

Prostřední část pod záložkami pak obsahuje formulář, kde uživatel vkládá jednotlivá data pro operace systému RKVAC. Jednotlivé vstupy formulářů pro každý protokol jsou odpovídající s aplikací RKVAC a údaje, které zde uživatel vyplní a odešle na server, jsou pak vstupem této aplikace. Nad formulářem jsou pak zobrazovány chybové hlášky, v případě, že je chod aplikace RKVAC neočekávaně přerušen. Napravo od formuláře je k dispozici také zaškrtávací políčko, kterým může uživatel ovládat, zda chce po provedení protokolu spustit odpovídající animaci nebo ne, jednotlivá tlačítka pro spuštění jednotlivých operací a protokolů nebo navigační tlačítko, které uživatele přesměruje na následující protokol. Téměř všechna tyto tlačítka (vyjma tlačítek, pomocí nichž lze zobrazovat výpis přístupů a revokovaných

uživatelů) můžou být deaktivovaná a to pokud není připojená čtečka karet a čipová karta nebo pokud zrovna probíhá již dříve spuštěná operace. Pod těmito tlačítky se nachází stavová ikonka, které má 4 stavy - výchozí stav (šedý otazník), indikace načítání a běhu operace, potvrzení o tom, že daná operace proběhla úspěšně (zelená fajfka) a nakonec indikace chyby nebo v případě ověření zamítnutí přístupu ke službě (červený křížek). Stavy ikon jsou zobrazeny na obrázku 4.9.

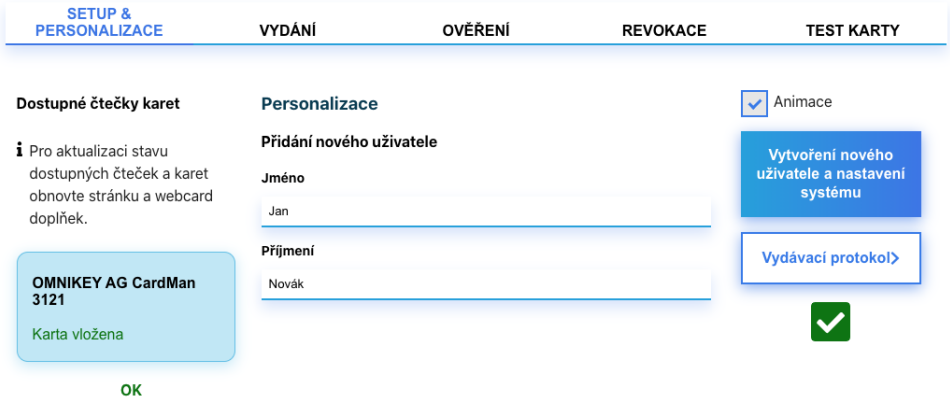


Obr. 4.9: Stavové ikony v kategorii Demo.

Formulář záložky *Setup a Personalizace* obsahuje dva textové vstupy pro jméno a příjmení uživatele (obrázek 4.10). Po zadání a odeslání dat na server (tlačítko *Vytvoření nového uživatele a nastavení systému*) je spuštěn v aplikaci RKVAC příkaz:

```
../rkvac-protocol-multos-1.0.0 -p -n jméno -s příjmení
```

a za předpokladu, že se nevyskytla chyba, je vytvořen nový uživatel a přidán záznam do souboru `user_list.csv` s odpovídajícím identifikátorem uživatele. Tlačítkem *Vydávací protokol* je uživatel přesměrován do záložky *Vydání*.

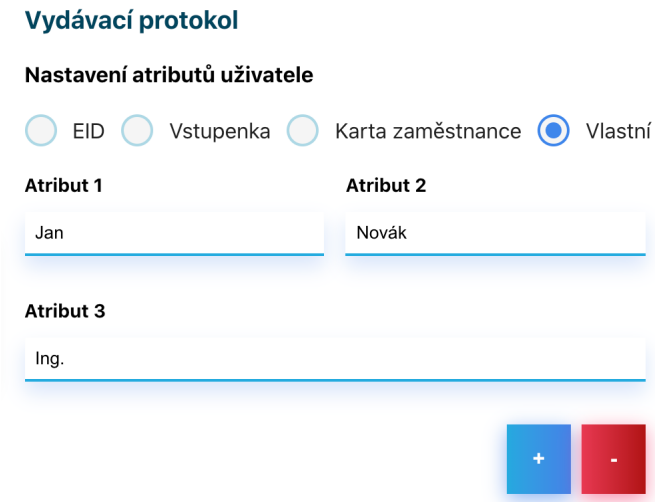


Obr. 4.10: Nastavení systému a personalizace uživatele.

V záložce *Vydání* (obrázek 4.12) jsou ve formuláři nejen textové vstupy, ale i přepínače, pomocí nichž si uživatel může zvolit jaký typ pověření chce od vydavatele získat. Zobrazené textové vstupy odpovídají aktivnímu přepínači a to:

1. **EID** - jméno a příjmení, datum narození, národnost, bydliště, pohlaví,
2. **vstupenka** - jméno a příjmení, číslo vstupenky, typ vstupenky,
3. **karta zaměstnance** - jméno a příjmení, id zaměstnance, zaměstnavatel, pozice zaměstnance,
4. **vlastní** - vlastní atributy, maximální počet atributů je 9.

V případě volby vlastních atributů má uživatel pomocí tlačítek + a - možnost se rozhodnout, kolik vlastních atributů bude na kartu vydáno. Formulář pro vydání pověření vlastních atributů je zobrazen na obrázku 4.11.



Obr. 4.11: Vydání pověření vlastních atributů.

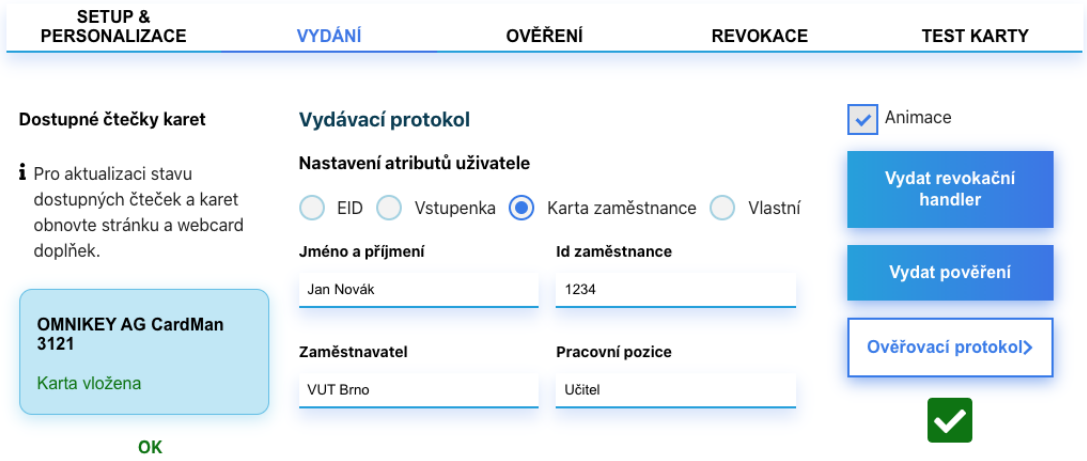
Před samotným vydáním pověření na kartu je nutné nejdříve na kartu zaslat data pro revokační handler (tlačítko *Vydat revokační handler*) spuštěním příkazu;

```
../rkvac-protocol-multos-1.0.0 -r
```

v RKVAC aplikaci. Vydání pověření na kartu v aplikaci RKVAC spouští na stránce tlačítko *Vydat pověření* a na pozadí příkaz;

```
../rkvac-protocol-multos-1.0.0 -i -a jménouživatele.att
```

Poslední zobrazené tlačítko *Ověřovací protokol* přesměruje uživatele do záložky *Ověření*.



Obr. 4.12: Vydání pověření.

Stejně jako záložka pro *Vydání*, tak i *Ověření* obsahuje formulář s přepínači odpovídajícím jednotlivým typům pověření a textové vstupy (obrázek 4.13). Každý tento vstup je pak doplněn i o zaškrtávací políčko, pomocí něhož uživatel ovládá, která konkrétní data (atributy) jsou ověřovateli odhalena. Před spuštěním ověření musí uživatel vytvořit novou epochu. Tento požadavek je podmíněn tím, že tlačítko pro ověření je neaktivní. Tlačítko *Vytvořit a aktivovat novou epochu* je nastaveno tak, že je možné vydat jednu epochu denně a pro pokračování dál je potřeba toto tlačítko stisknout podruhé. Pokud již byla pro daný den vydána, je o tom uživatel informován a je zobrazeno doplňující tlačítko *Vytvořit další*, které umožňuje ve stejném dni vydat i další epochy. Toto doplňující tlačítko pro další epochy neovlivňuje stav tlačítka pro ověření.

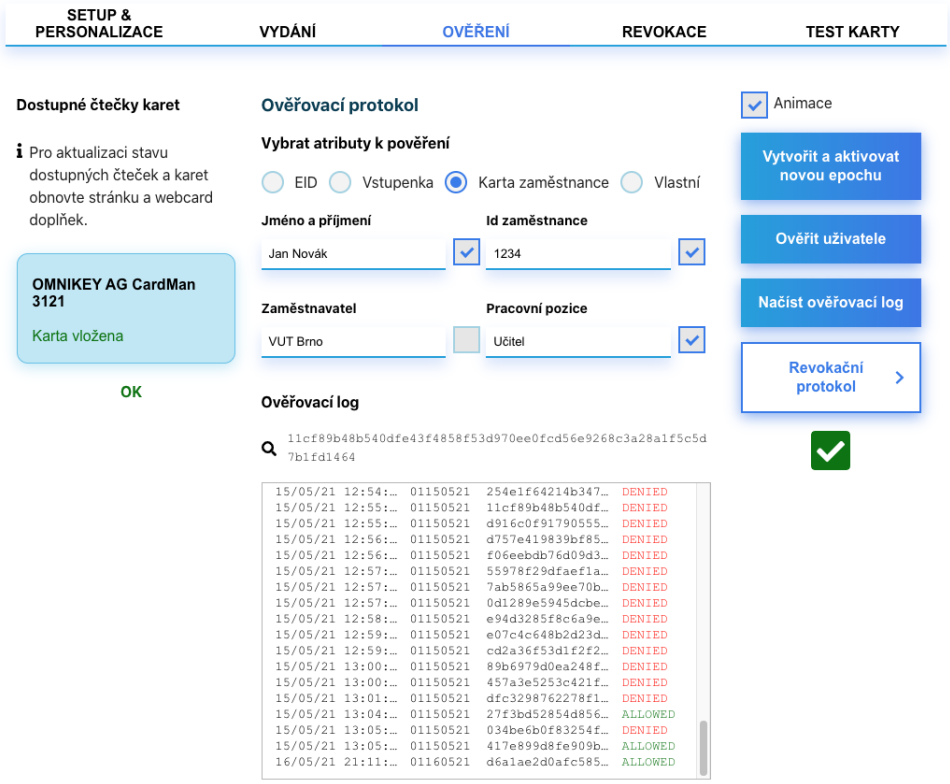
Ověření uživatele se provede stiskem tlačítka *Ověřit uživatele* – na straně serveru je spuštěn příkaz;

```
../rkvac-protocol-multos-1.0.0 -v -a
    jménouživatele\_timestamp.att
```

Aplikace RKVAC je nakonfigurována tak, že v případě, kdy pod stejným názvem souboru již dříve probíhalo ověření, jsou načtena data z tohoto běhu a ověřovatel tak pracuje s těmito starými daty (přestože ve skutečnosti jsou vstupní data rozdílná). Z tohoto důvodu je vždy vytvořen nový soubor, který v názvu obsahuje hodnotu časového razítka (*timestampu*).

Pod celým formulářem se pak nachází tabulka *Ověřovací log*, která zobrazuje data o provedených ověřovacích operacích. Tato tabulka obsahuje informace o datu a času vytvoření ověření, číslo epochy, ve které bylo ověření prováděno, pseudonym a také informaci o tom, zda bylo ověření validní nebo ne, tzn. jestli byl uživateli udělen

přístup. Zpravidla údaje pro datum, čas a pseudonym mají větší počet znaků, proto jsou v tabulce ve zkráceném tvaru a nad ní se nachází prostor pro výpis těchto dat, která se zobrazí po přejetí kurzoru na danou hodnotu v tabulce. Tato tabulka je zobrazena a aktualizována každým spuštěním ověřovací operace nebo může být nezávisle na ověření zobrazena stiskem tlačítka *Načíst ověřovací log*.



Obr. 4.13: Ověření uživatele.

Stejně jako v předchozích případech, i zde je doplňující tlačítko *Revokační protokol* pro přesměrování uživatele.

Revokace obsahuje seznam uživatelů, kteří byli do systému inicializování – u každého uživatele je uvedeno jeho jméno, příjmení a ID a seznam pseudonymů. Každý pseudonym je doplněn o údaj, ve které epoše byl vytvořen a také, jestli je k němu vázáno validní nebo nevalidní ověření. Tyto seznamy a výpis již revokovaných uživatelů lze zobrazit stiskem tlačítka *Načíst seznam uživatelů a pseudonymů*. Z těchto dvou seznamů si může uživatel vybrat, zda chce provést revokaci na základě uživatele nebo pseudonymu. Tlačítko *Revokovat* je ekvivalentní k příkazu

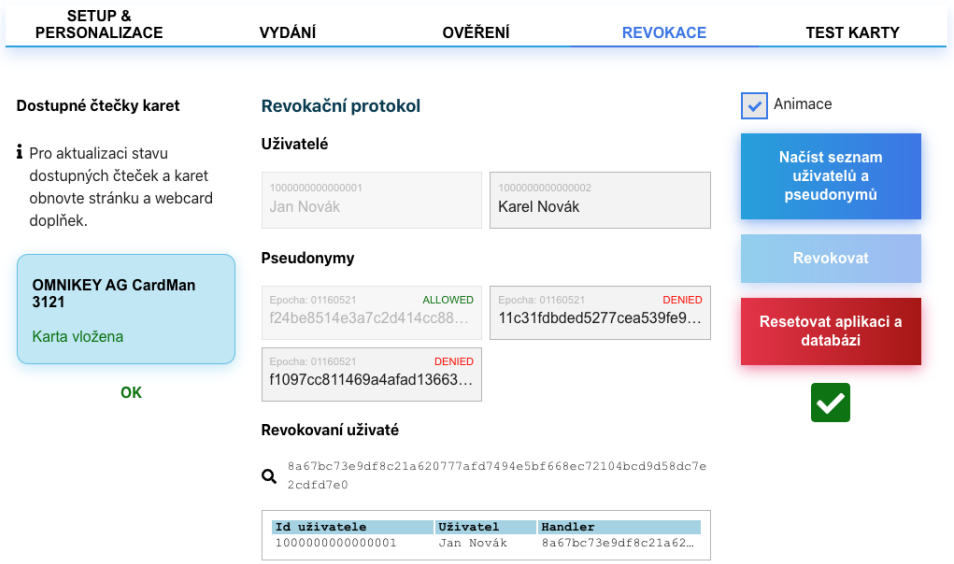
```
../rkvac-protocol-multos-1.0.0 -r -B 'epocha␣iduživatele'
```

nebo

```
../rkvac-protocol-multos-1.0.0 -r -b 'epocha␣pseudonym'
```

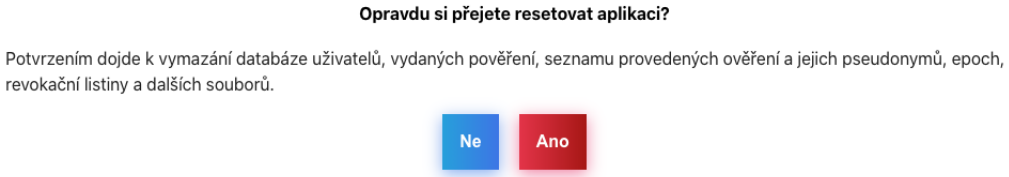
podle toho, zda je revokace provedena na základě ID uživatele nebo pseudonymu C.

Pod tímto seznamem nachází tabulka, která obsahuje seznam revokovaných uživatelů – jejich ID, jméno a příjmení a revokační handler, který se k danému uživateli váže. Data, která obsahují větší počet znaků je opět možné zobrazit nad tabulkou přejetím kurzoru v tabulce. Záložka Revokace je zobrazena na obrázku 4.14.



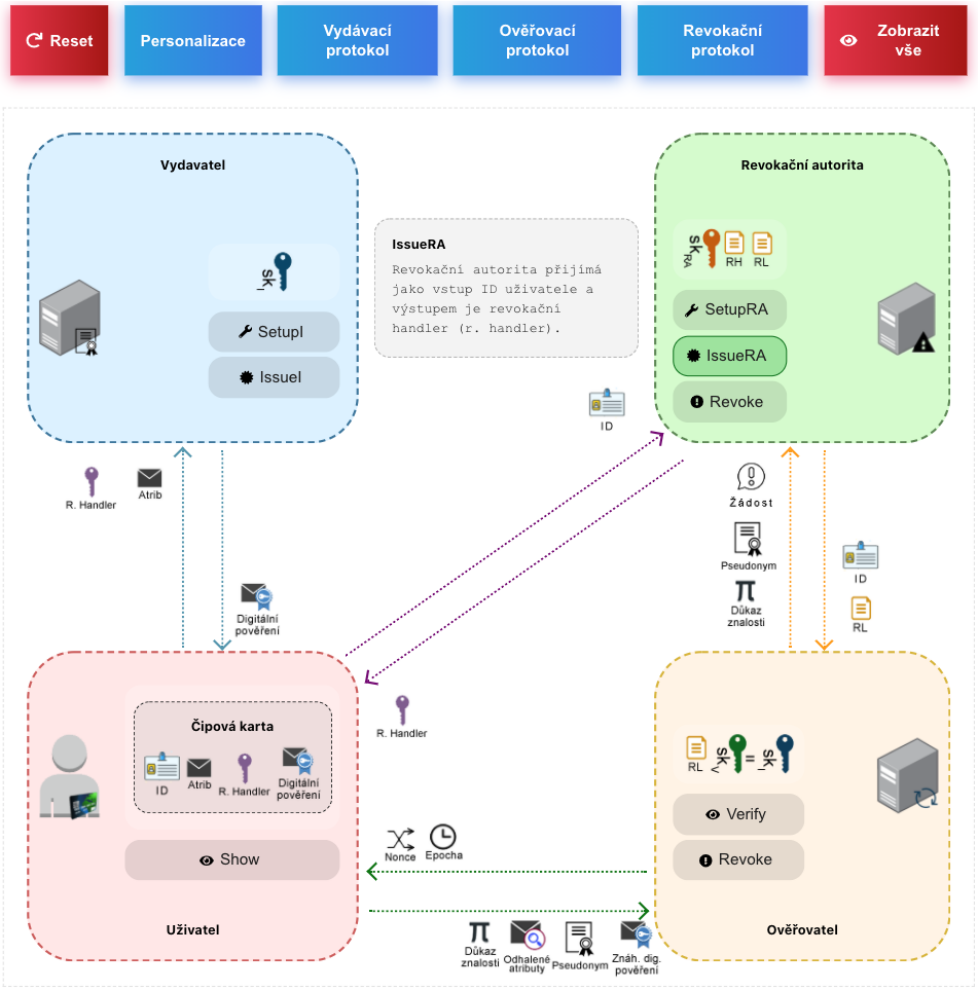
Obr. 4.14: Revokace uživatele.

V této sekci je také možné resetovat celou aplikaci a vymazat databázi, která obsahuje data pro pověření, ověření, všechny revokační listiny, epochy apod. Reset může být proveden tlačítkem *Resetovat aplikaci a databázi*. Po kliknutí na toto tlačítko se zobrazí vyskakovací okno (obrázek 4.15), kdy je uživatel znovu dotázán, zda si opravdu přeje tuto operaci provést. To zamezuje nechtěnému vymazání dat v případě, kdy uživatel stiskne původní tlačítko omylem.



Obr. 4.15: Reset aplikace a vymazání databáze.

Druhá část kategorie Demo je vyhrazena pro animovanou vizualizaci systému RKVAC (obrázek 4.16). Tato vizualizace je dostupná pouze pro šířku zobrazení 1060 pixelů a více. Při menším rozlišení by jednotlivé části a ikonky ve vizualizaci byly malé a špatně čitelné. Animace v této vizualizaci jsou spuštěny buď zároveň s během demonstrátoru výše popisovaného nebo nezávisle na tomto demonstrátoru a to i bez připojené čtečky karet a čipové karty. Ty lze spouštět manuálně stiskem jednotlivých tlačítek umístěných nad vizualizací. Kromě tlačítek s názvem jednotlivých protokolů se v této části nachází i tlačítko *Reset*, čímž se celá animace přepne do výchozího stavu, nebo *Zobrazit vše*, kdy si uživatel může zobrazit schéma systému RKVAC a komunikaci mezi jednotlivými entitami jako jeden celek. Kromě entit a znázorněné komunikace mezi nimi v rámci jednotlivých protokolů, vizualizace také obsahuje informační bublinu s názvem a krátkým popisem právě animovaného protokolu. V této bublině lze také zobrazovat krátké popisy jednotlivých algoritmů a to stiskem tlačítek s názvem algoritmu, která jsou umístěna v entitách.



Obr. 4.16: Část kategorie Demo – animace.

4.3 Instalace na OS Linux

Instalace na operační systém Linux je zaměřena zejména na server. Na tomto OS lze spustit i klientskou část, ale přístup k webové stránce, kterou klient poskytuje, musí být ze vzdálené stanice s operačním systémem Windows nebo MacOS a nainstalovaných doplňkem Webcard.

Serverová část je rozdělená na dvě podčásti. První je server napsán v NodeJS, který komunikuje s klientem a druhá je RKVAC aplikace v jazyce C. Naopak na tomto OS není možné zprovoznit klientskou část, protože instalace doplňku Webcard je omezená na OS Windows a MacOS.

Tato serverová část byla v rámci této práce řádně otestována na linuxové distribuci Ubuntu ve verzi 20.04, proto i návod na spuštění odpovídá této distribuci a u jiných může být postup odlišný. Instalace na Linux je podmíněná RKVAC aplikací, která je navržena na architekturu tohoto operačního systému.

4.3.1 RKVAC aplikace

Podrobný popis instalace této aplikace je nad rámec této práce. Návod pro instalaci je přiložen ve složce, kde se nachází instalační soubory pro tuto aplikaci (složka `rkvac`).

4.3.2 Server

Pro spuštění serveru je nezbytné mít na stanici nainstalovaný nástroj NodeJS ve verzi 10 a vyšší. Instalace na Ubuntu se provede příkazem:

```
sudo apt install nodejs
```

Tento příkaz nainstaluje na stanici poslední stabilní verzi nástroje NodeJS. V dalším kroku je důležité celou složku `server` přesunout do složky `build`, která se nachází v adresáři `rkvac` (po buildu aplikace RKVAC), cesta k serverové složce tedy bude:

```
/rkvac/build/server
```

Dále je potřeba nainstalovat související balíčky, které jsou definované v souboru `package.json`. V adresáři, kde se soubory pro server nachází (`/server`), stačí zadat následující příkaz:

```
npm install
```

Po úspěšné instalaci všech závislostí už jen zbývá server spustit příkazem:

```
npm start
```

Běžící server je nyní schopen poslouchat na portu 3002, od klienta přijímat požadavky a zároveň na portu 3001 otevírá komunikaci přes WebSocket.

4.3.3 Webová aplikace

Na OS Linux lze spustit klientskou část aplikace, tzn. webovou stránku. Protože však na Linuxu není zajištěná 100% funkčnost (pro komunikaci s čipovou kartou a přenos dat je nutné mít v prohlížeči doplněk Webcard, který nelze nainstalovat na OS Linux), musí uživatel k tomuto běžícímu klientovi přistupovat vzdáleně přes síť a IP adresu stanice.

Protože už je na stanici nainstalován nástroj NodeJS z předchozího kroku, instalace a spuštění klienta zahrnuje dva příkazy. V prvním kroku je ve složce *klient* potřeba nainstalovat balíčky:

```
npm install
```

V dalším kroku už jen stačí spustit server pro klientskou část:

```
npm start
```

V tomto případě, kdy jede klientská i serverová část na jedné stanici a je k ní přistupováno vzdáleně, je potřeba upravit IP adresy v souborech `package.json` a `api.js` (stejně jako u instalace na OS Windows nebo MacOS, viz kapitola 4.4 a kapitola 4.5). Ve výchozím nastavení těchto dvou souborů není konkrétní adresa, ale pouze obecně `localhost`.

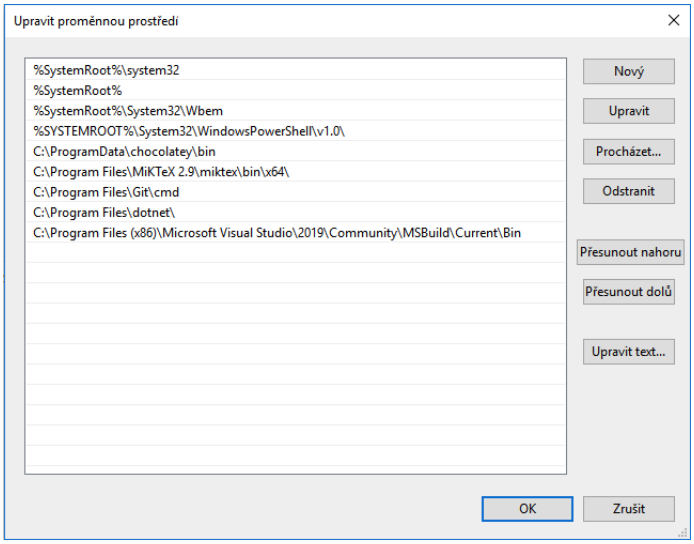
4.4 Instalace na Windows

Webová aplikace na operačním systému Windows vyžaduje instalaci několika programů a balíčků. Protože serverová část komunikuje s aplikací RKVAC, která musí být nainstalovaná lokálně a tato aplikace je funkční pouze v prostředí Linux, je v této části popsána pouze instalace webové aplikace (klienta) a doplňku Webcard.

Pro běh webové aplikace je zapotřebí instalace programu NodeJS. Tento program je pro operační systém Windows k dispozici ke stažení na oficiálních stránkách jako spustitelný instalační soubor [26]. Požadovaná verze NodeJS pro správný chod aplikace je verze 10 a vyšší. Po instalaci je doporučeno nastavit v systému proměnnou prostředí, tj. cestu ke spustitelnému binárnímu souboru.

4.4.1 Nastavení proměnné prostředí

Po kliknutí pravým tlačítkem myší na ikonu `Tento počítač` je z kontextového menu zvolena možnost `Vlastnosti`. Dále pak ze sloupce dalších nabídek možnost `Upřesnit nastavení systému`, karta `Upřesnit` a možnost `Proměnné prostředí`. V následujícím okně je ze skupiny `Systémové proměnné` zvolena proměnná `Path` a po kliknutí na tlačítko `Upravit...` je otevřeno další okno. Po stisknutí tlačítka `Nový` je pak posledním krokem vložení absolutní cesty spustitelného souboru do textového vstupu (obrázek 4.17). Po uložení všech provedených změn je možno v terminálu používat příkaz, který se váže ke konkrétnímu souboru, např. `node`, `npm`, apod...



Obr. 4.17: Nastavení proměnné prostředí v OS Windows.

V případě, že není nastavena tato cesta jako proměnná prostředí, je možné v terminálu volat příkazy skrz absolutní cestu, která vede ke spustitelnému souboru (např. `"C:\Program Files\nodejs\node"`) .

4.4.2 Webová aplikace

Pro běh webové aplikace je zapotřebí instalace potřebných závislostí. Všechny tyto závislé balíčky jsou definované v souboru `package.json` v adresáři `/client`. Balíčky se nainstalují pomocí příkazu

```
npm install
```

Spuštěním tohoto příkazu je vytvořen adresář `node_modules`, který tyto specifikované balíčky obsahuje. Instalace může chvíli trvat. Následně je příkazem

```
npm start
```

spuštěna webová aplikace, která běží na portu 3000. Přístup k aplikaci, která běží lokálně, je v prohlížeči přes `localhost:3000`.

V dalším kroku je potřeba nakonfigurovat adresu serveru, na který bude klient posílat data. Ve výchozím nastavení je v souborech nastaven localhost. Protože však nelze serverovou část nainstalovat na stejnou stanici, kde se nachází webová aplikace a doplněk Webcard (aplikaci RKVAC, která je součástí serveru, nelze spustit na OS Windows), je potřeba nastavit IP adresu. Adresa serveru bude nakonfigurována v souborech (číslo portu může být zachováno):

- `package.json` – řádek 7, položka proxy, adresa a port pro API,
- `api.js` – řádek 5, adresa pro WebSocket spojení.
  Cesta k souboru je `src/api/api.js`.

4.4.3 Webcard

Webcard v prostředí Windows je možné nainstalovat dvěma způsoby. První způsob je instalace přes tzv. Internetový obchod Chrome, kde lze jednoduše do prohlížeče instalovat doplňky, motivy a jiná rozšíření (toto řešení by pokrývalo jak OS Windows, tak i MacOS a případně další operační systémy). Instalace doplňku Webcard je tímto způsobem velmi snadná a rychlá, ale doplněk je blokován pouze pro použití na demo stránce Webcardu – `webcard.cardid.org`. S cílem co největšího usnadnění instalace tohoto doplňku byl prostřednictvím e-mailové komunikace kontaktován jeho hlavní vývojář. Dle informací obdržených od vývojáře, tento typ doplňku, který je zveřejněný na Internetovém obchodu Chrome a posléze z tohoto obchodu nainstalován, nemůže z bezpečnostních důvodů běžet na jiných než explicitně specifikovaných webech. Tímto by byl však chod aplikace v této práci značně omezen a z tohoto důvodu je nutné Webcard nainstalovat druhým způsobem.

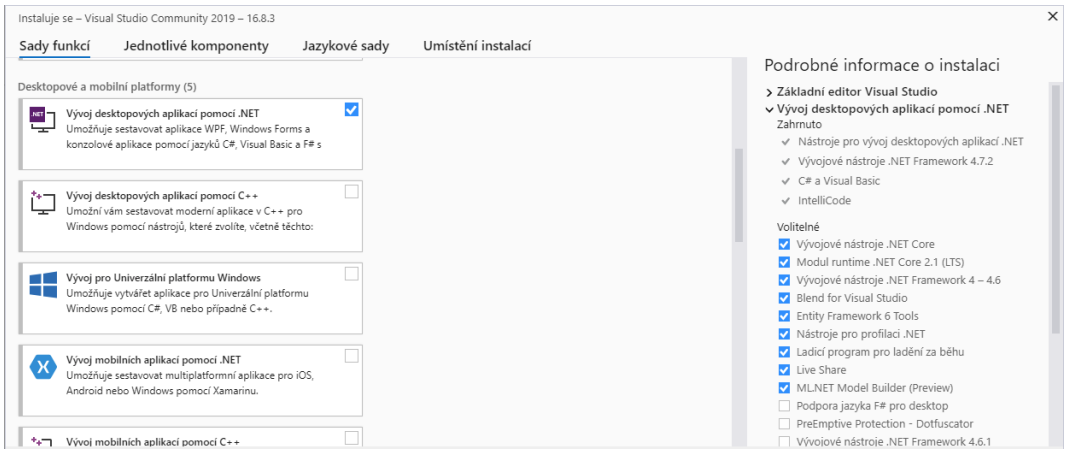
Webcard v prostředí OS Windows mimo instalaci samotného rozšíření do prohlížeče Chrome vyžaduje kompilaci několika programů napsaných v jazyce C++ a následně instalaci přidružených souborů.

V první fázi je vyžadována instalace doplňku do prohlížeče Chrome. V kategorii `Nástroje -> Rozšíření` je nutné zapnout `Režim pro vývojáře` a přes možnost `Načíst nerozbalené` nahrát složku `extension`, která se nachází v adresáři `/webcard`. Doplněk má přiřazený náhodný klíč. Tento klíč bude potřeba vložit do souborů v následujících krocích.

Nástroj MSBuild

V další fázi je potřeba provést kompilaci a sestavit instalační soubor pro aplikaci, která komunikuje s čipovou kartou. To je provedeno pomocí nástroje MSBuild, který je v rámci programu Microsoft Visual Studio volně dostupný ke stažení

[30]. V instalačním programu je vyžadováno nainstalovat balíčky nástrojů `Vývoj desktopových aplikací pomocí .NET`, který obsahuje nástroj MSBuild a balíček `Vývoj desktopových aplikací pomocí C++`, který je také stěžejním pro úspěšnou kompilaci (obrázek 4.18).



Obr. 4.18: Instalační prostředí Microsoft Visual Studio.

Po úspěšné instalaci lze nadefinovat cestu (viz kapitola 4.4.1) a poté ve složce `native` spustit příkaz;

```
MSBuild webcard.vcxproj -p:Configuration=Release
```

Tento příkaz vygeneruje složku `Release`. Obsah souboru `webcard.wxs` ve složce `native` je pro další postup nezbytné upravit a to následovně:

1. **Řádek 35, 36, 37** – výrazy obsahující řetězec `org.cardid.webcard` nahradit řetězcem `org.cardid.webcard.native` (viz výpis 6),
2. **Řádek 41** – přepsat klíč doplňku pro Chrome a to ve tvaru: `Software\Google\Chrome\Extensions\<klíč>`. Hodnota klíče byla dříve vygenerována a je dostupná v prohlížeči v kategorii `Rozšíření`.

Dále je potřeba v souboru `org.cardid.webcard.native.json` na řádku 4 upravit cestu ke spustitelné aplikaci: `"path": "webcard.exe",` a nahradit původní identifikační klíč doplňku podle nově vygenerovaného (řádek 7). Úprava souboru je zobrazena ve výpise 6.

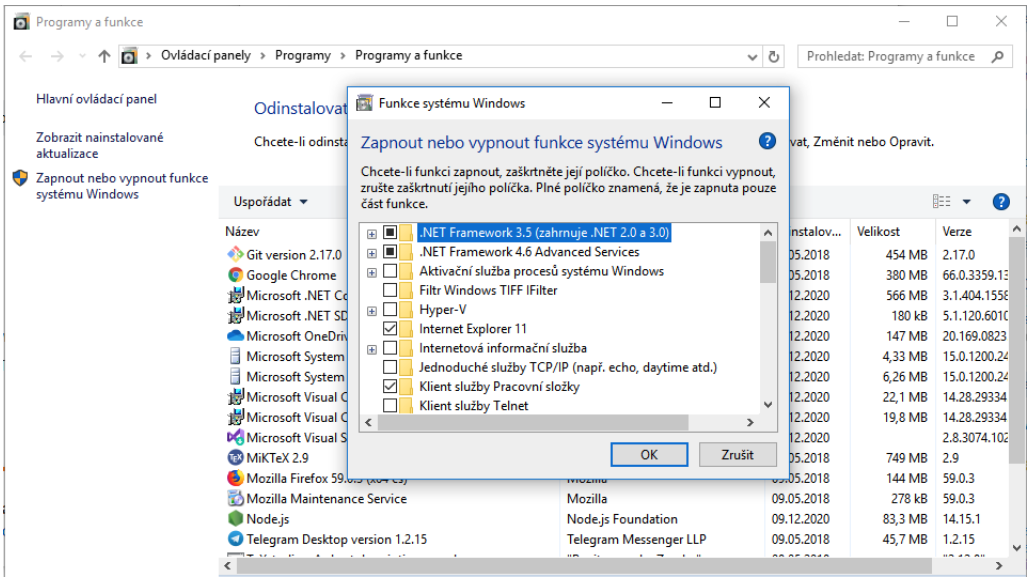
```
<File Id="WebcardJSON" Name="org.cardid.webcard.native.json"
 ↪  DiskId="1" Source="org.cardid.webcard.native.json" />
<RegistryKey Id="WebcardJsonReg" Root="HKCU"
 ↪  Key="Software\Google\Chrome\NativeMessagingHosts\
  org.cardid.webcard.native" ForceDeleteOnUninstall="yes">
<RegistryValue Type="string"
 ↪  Value="[INSTALLDIR]\org.cardid.webcard.native.json"
 ↪  KeyPath="yes" />
```

Výpis 6: Soubor webcard.wxs - úprava obsahu souboru.

Nástroj WiX toolset

Pro následující postup je pomocí nástroje WiX toolset [35], který umožňuje vytvářet instalační soubory pro OS Windows, vytvořen instalační soubor pro aplikaci Webcard. WiX toolset je volně dostupný a jeho instalační soubor se nachází na oficiálních stránkách tohoto nástroje.

Instalace tohoto nástroje vyžaduje zapnutou funkci aplikace `.NET Framework 3.5`. To lze zkontrolovat, případně zapnout tímto způsobem: `Ovládací panely`, `Programy`,
`Programy a funkce`, možnost `Zapnout nebo vypnout funkce systému Windows` z nabídky v levém sloupci. Po otevření okna pak zaškrtnout položku `.NET Framework 3.5 (zahrnuje .NET 2.0 a 3.0)`



Obr. 4.19: Funkce .NET Framework.

Po tomto nastavení, případně kontrole, by měl jít WiX toolset přes jeho instalační soubor bezproblémově nainstalovat. Po instalaci by v terminálu měl být nově dostupný příkaz `candle.exe` a `light.exe`.

Dalším krokem je již vytvoření instalačního souboru `webcard.exe`. Výchozím souborem je dříve upravený soubor `webcard.wxs`. V terminálu je po zadání příkazu

```
candle.exe webcard.wxs
```

ve složce `/native` vygenerován soubor `webcard.wixobj` a po zadání příkazu

```
light.exe webcard.wixobj
```

je ve stejné složce dostupný instalátor `webcard.msi`. Po spuštění by měla být dokončena instalace aplikace webcard v systému. Pro správnou funkčnost stačí obnovit související doplněk v Chrome a komunikace webové stránky s čipovou kartou přes webové rozhraní by měla být plně funkční.

4.5 Instalace na MacOS

Instalace aplikace je, stejně jako u OS Windows, rozdělena do 2 částí – webová aplikace a doplněk Webcard. Protože tyto dva systémy jsou po stránce jejich architektury rozdílné, je postup pro instalaci jednotlivých balíčků, knihoven a aplikaci mírně odlišný. Opět jako u OS systému Windows je i zde vynechána část instalace serveru, který musí běžet na OS Linux.

4.5.1 Webová aplikace

Pro spuštění klientské části je požadována instalace nástroje NodeJS. Nástroj může být nainstalován vícero způsoby, z nichž dva jsou následující:

- Instalace pomocí instalačního souboru, který lze stáhnout na oficiálních stránkach NodeJS. Instalace je podobná jako u OS Windows.
- Instalace přes terminál a nástroj Homebrew [36]. Homebrew je velmi populární balíčkovací manažer pro operační systém MacOS, pomocí něhož lze instalovat další nástroje jen za použití terminálu. Instalace Homebrew je snadná a rychlá. NodeJS je pomocí Homebrew nainstalován jedním příkazem a to:

  ```
  brew install node
  ```

Požadovaná verze NodeJS pro správný chod aplikace je verze 10 a vyšší. Následující postup je již shodný s OS Windows. Po instalaci NodeJS je potřeba instalace balíčků, které se nachází ve složce `client`;

```
npm install
```

Stejně jako u OS Windows (kapitola 4.4.2), i na MacOS je potřeba změnit adresu serveru pro zasílání dat z výchozího `localhost`. Adresa serveru bude nakonfigurována v souborech (číslo portu může být zachováno):

- `package.json` – řádek 7, položka proxy, adresa a port pro API,
- `api.js` – řádek 5, adresa pro WebSocket spojení.

Spuštění klienta se pak provede příkazem:

```
npm start
```

4.5.2 Webcard

Prvním krokem je instalace doplňku do prohlížeče Chrome. Postup je shodný již se zmíněnou instalaci u OS Windows. V kategorii `Nástroje -> Rozšíření` v zapnutém módu `Režim pro vývojáře` je potřeba načíst složku s doplňkem `extension` z adresáře `/webcard` přes možnost `Načíst nerozbalené`. Náhodný klíč, který je doplňku přiřazen se bude dál využívat pro následující kroky.

Instalace aplikace pro Webcard do počítače je na MacOS oproti systému Windows snadná a rychlá. Před samotnou instalaci je potřeba upravit následující soubory:

1. `install.sh` – na řádku 28 je potřeba upravit hodnotu `EXTENSION_ID` ID hodnotou doplňku webcard vygenerovanou v prohlížeči,
2. `org.cardid.webcard.native.json` – úprava řádku 7. Upravit hodnotu mezi lomítky opět ID hodnotou doplňku webcard.

Další kroky jsou provedeny přes terminál `iTerm` (případně přes jinou aplikaci stejného typu). Ve složce `/native` se spuštěním příkazu `make` zkompilují zdrojové soubory. Dále po provedení příkazu

```
./install.sh
```

je aplikace nainstalována do počítače. Pokud byly tyto kroky úspěšně provedeny, měl by být doplněk webcard úspěšně nainstalován a funkční. V případě, že i po úspěšné instalaci doplněk webcard v prostředí prohlížeče Chrome nefunguje, je potřeba prohlížeč restartovat a dále pak restartovat (nezřídka i několikrát po sobě) samotný doplněk v sekci Rozšíření.

Závěr

Hlavním cílem této práce byla implementace interaktivní webové aplikace, která vizualizuje funkčnost anonymního pověřovacího schématu s revokací RKVAC. Záměrem bylo, aby tato aplikace byla schopná komunikovat se čtečkou čipových karet a čipovou kartou a zajistila tak přenos dat. Aplikace je kompatibilní s chytrými telefony s podporou NFC a nainstalovanou RKVAC aplikací. Aplikace je rozdělena na dva větší celky, které jsou navrženy jako klient a server.

Server obsahuje dvě části. První část je implementována v NodeJS a funguje jako komunikační prostředník mezi klientem a poskytnutou RKVAC aplikací napsanou v jazyce C, která poskytuje výpočty protokolů a sestavení APDU zpráv určených pro čipovou kartu. Od klienta přijímá požadavky, na jejichž základě provádí operace a spouští RKVAC aplikaci a také zajišťuje přenos dat – APDU příkazů a odpovědí. Mezi klientem a serverem jsou dva komunikační kanály, první je API rozhraní, kterým si mezi sebou klient a server zasílají požadavky a odpovědi, druhým je WebSocket určený pro přenos APDU.

Klientská část zahrnuje webovou stránku, ke které uživatel přistupuje přes webový prohlížeč Google Chrome. Použití tohoto prohlížeče je podmíněné doplňkem Webcard, který zajišťuje přenos APDU zpráv mezi webovou aplikací a čipovou kartou MultOS. Webová stránka zahrnuje 4 hlavní kategorie – Popis, Entity, Protokoly a Demo. Kategorie *Popis* obsahuje obecný popis schématu RKVAC, kategorie *Entity* popis jednotlivých entit a jejich funkcí, kategorie *Protokoly* obsahuje výčet a popis protokolů, které se v tomto schématu nachází a kategorie *Demo* obsahuje demonstrátor funkčnosti protokolu. Tato kategorie je vizuálně rozdělena na dvě části. První část je vyhrazena pro komunikaci se serverovou částí – obsahuje záložky pro jednotlivé protokoly, seznam připojených čteček čipových karet a dále formuláře, které strukturou odpovídají RKVAC aplikaci a jejím vstupním parametrům. Odesíláním dat z formulářů se zároveň na straně serveru spouští aplikace RKVAC s parametry, které odpovídají údajům z formuláře z webové stránky. Záložka *Ověření* je doplněná o výpis dat z ověřovacího logu a záložka *Revokace* obsahuje seznam revokovaných uživatelů a tlačítko pro smazání všech dat z aplikace. Mezi serverem a klientem se otevírá spojení pro komunikaci přes WebSocket a je tak zajištěn reálný přenos APDU příkazů a odpovědí mezi RKVAC aplikací a čipovou kartou. Druhá část kategorie *Demo* obsahuje vizualizaci. Tato vizualizace je buď spouštěna automaticky za běhu jednotlivých protokolů nebo manuálně bez ohledu na to, zda je karta připojena nebo spuštěn server. Nad vizualizací se nachází tlačítka pro simulaci jednotlivých protokolů, kterými lze animace spouštět. Animace je doplněna o popis jednotlivých protokolů nebo algoritmů a přepínat mezi těmito popisy lze tlačítky umístěnými přímo ve vizualizaci.

Implementace této práce je zaměřena na programovací jazyk JavaScript a jeho dostupné nástroje a frameworky. Na straně serveru je použit nástroj NodeJS s doplňující knihovnou Express.js, klient je postaven na populárním frameworku ReactJS a dalšími podpůrnými balíky.

Aplikace funguje jako nástroj, který uživateli rozšíří povědomí o schématu RKVAC a jeho principech a také jako interaktivní grafické uživatelské rozhraní, doplněné o vizualizaci, pomocí něhož je uživatel schopen snadno komunikovat s aplikací pro anonymní pověřovací schéma s revokací (RKVAC). Aplikace je taktéž schopna zajistit vydání pověření na připojenou čipovou kartu nebo z čipové karty odesílat pověření v rámci ověřovacího protokolu a prokázat se tak u ověřovací autority.

Literatura

[1] Microsoft Research: *U-Prove* [online]. [cit. 2020-12-10]. Dostupné z URL: <https://www.microsoft.com/en-us/research/project/u-prove/>.

[2] FEIGE U., FIAT A., SHAMIR A.: *Zero-Knowledge Proofs of Identity.* [online]. [cit. 2020-12-10]. Dostupné z: <http://crypto.cs.mcgill.ca/~crepeau/COMP647/2010/TOPIC03/FFS88.pdf>.

[3] *Schnorr's Protocol.* [online]. [cit. 2020-12-10]. Dostupné z: <http://www.lsv.fr/Software/spore/schnorr.pdf>.

[4] OCHODKOVÁ, Eliška: *Matematické základy kryptografických algoritmů* [online]. [cit. 2021-22-05]. Dostupné z URL: <http://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/mat_zaklady_kryptografickych_algoritmu.pdf>.

[5] BARKER E.: *Recommendation for Key Management: Part 1 – General* [online]. [cit. 2021-20-03]. DOI: 10.6028/NIST.SP.800-57pt1r5. Dostupné z URL: <https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-57pt1r5.pdf>.

[6] ZEMAN K., MAŠEK P., HOŠEK J.: *Experimentální ověření vhodnosti nízkoodběrových IoT zařízení pro implementaci kryptografických primitiv.* Elektrorevue, 2016, č. 1, s. 18. [online]. [cit. 2020-12-01]. Dostupné z: <http://www.elektrorevue.cz/cz/download/experimentalni-overeni-vhodnosti-nizkoodberovych-iot-zarizeni-pro-implementaci-kryptografickych-primitiv--experimental-verification-of-suitability-of-low-power-devices-for-implementation-of-cryptographic-primitives-/>.

[7] MORAVANSKÝ, Michal. Implementace kryptografických protokolů na čipové karty [online]. Brno, 2020 [cit. 2020-12-10]. Dostupné z: <http://hdl.handle.net/11012/190291>. Bakalářská práce. Vysoké učení technické v Brně. Fakulta elektrotechniky a komunikačních technologií. Ústav telekomunikací. Vedoucí práce Petr Dzurenda.

[8] CAMENISCH J.: *Concepts Around Privacy-Preserving Attribute-Based Credentials.* [online]. [cit. 2020-12-10]. Dostupné z: <https://hal.archives-ouvertes.fr/hal-01276046/document>.

[9] Hyperledger Fabric: *MSP Implementation with Identity Mixer* [online]. [cit. 2020-12-10]. Dostupné z URL: <https://hyperledger-fabric.readthedocs.io/en/release-1.3/idemix.html#what-is-idemix>.

[10] CAMENISCH, Jan, Manu DRIJVERS, Petr DZURENDA and Jan HAJNY. *Fast Keyed-Verification Anonymous Credentials on Standard Smart Cards.* In *ICT Systems Security and Privacy Protection* [online]. Springer Nature Switzerland, 2019, s. 1-13 [cit. 2020-12-10]. ISBN: 978-3-030-22312-0. Dostupné z URL: <http://link.springer.com/10.1007/978-3-030-22312-0_20>.

[11] The WebSocket API (WebSockets) [online]. [cit. 2021-05-08]. Dostupné z URL: <https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API/>.

[12] Node.js [online]. [cit. 2020-12-09]. Dostupné z URL: <https://nodejs.org/en/>.

[13] npm: *Build amazing things* [online]. [cit. 2020-12-10]. Dostupné z URL: <https://www.npmjs.com/>.

[14] Express.js: *Node.js web application framework* [online]. [cit. 2020-12-10]. Dostupné z URL: <https://expressjs.com/>.

[15] Moment.js [online]. [cit. 2021-05-16]. Dostupné z URL: <https://momentjs.com/>.

[16] csv-parser - npm [online]. [cit. 2021-05-16]. Dostupné z URL: <https://www.npmjs.com/package/csv-parser>.

[17] React: *A JavaScript library for building user interfaces* [online]. [cit. 2020-12-01]. Dostupné z URL: <https://reactjs.org/>.

[18] styled-components: *Visual primitives for the component age* [online]. [cit. 2020-12-01]. Dostupné z URL: <https://styled-components.com/>.

[19] GreenSock: *GSAP* [online]. [cit. 2020-12-10]. Dostupné z URL: <https://greensock.com/gsap/>.

[20] i18next documentation: Introduction [online]. [cit. 2021-05-16]. Dostupné z URL: <https://www.i18next.com>.

[21] Prettier – Opinionated Code Formatter [online]. [cit. 2021-05-16]. Dostupné z URL: <https://prettier.io>.

[22] ESLint – Pluggable JavaScript linter [online]. [cit. 2021-05-16]. Dostupné z URL: <https://eslint.org>.

[23] The Chromium Projects: *NPAPI deprecation: developer guide* [online]. [cit. 2020-12-01]. Dostupné z URL: <http://www.chromium.org/developers/npapi-deprecation>.

[24] Python 3.9.1: *Subprocess management* [online]. [cit. 2020-12-11]. Dostupné z URL: <https://docs.python.org/3/library/subprocess.html>.

[25] npm: *pcsclite* [online]. [cit. 2020-12-11]. Dostupné z URL: <https://www.npmjs.com/package/pcsclite>.

[26] Node.js: *Download* [online]. [cit. 2020-12-09]. Dostupné z URL: <https://nodejs.org/en/download/>.

[27] Stack Overflow: *WebUSB: The requested interface implements a protected class* [online]. [cit. 2020-12-01]. Dostupné z URL: <https://stackoverflow.com/a/54913997>.

[28] Smart Card Connector [online]. [cit. 2020-12-01]. Dostupné z URL: <https://chrome.google.com/webstore/detail/smart-card-connector/khpfeaanjngmcnplbdlpegiifgpfgdco>.

[29] ROUSSEAU L.: *PC/SC sample in Smart Card Connector on Chromebook* [online]. Dostupné z URL: <https://ludovicrousseau.blogspot.com/2017/03/pcsc-sample-in-smart-card-connector-on.html>.

[30] Microsoft Docs: *Použití nástroje MSBuild - Visual Studio* [online]. [cit. 2020-12-09]. Dostupné z: <https://docs.microsoft.com/cs-cz/visualstudio/msbuild/walkthrough-using-msbuild?view=vs-2019>.

[31] webcard: *Smart Card Browser Extension* [online]. [cit. 2020-12-01]. Dostupné z: <https://github.com/cardid/webcard>.

[32] Getting started tutorial [online]. [cit. 2020-12-01]. Dostupné z URL: <https://developer.chrome.com/extensions/getstarted>.

[33] DOM Standard [online]. [cit. 2021-05-21]. Dostupné z URL: <https://dom.spec.whatwg.org/>.

[34] Native messaging [online]. [cit. 2020-12-01]. Dostupné z URL: <https://developer.chrome.com/extensions/nativeMessaging>.

[35] WiX Toolset [online]. [cit. 2020-12-10]. Dostupné z: <https://wixtoolset.org/releases/>.

[36] Homebrew – Chybějící správce balíků pro macOS [online]. [cit. 2021-05-16]. Dostupné z: <https://brew.sh/>.

Seznam symbolů, veličin a zkratek

AAA	Authentication Authorization and Accounting
ABC	Atributové pověření – Attribute-Based Credential
APDU	Application Protocol Data Unit
API	Application Programming Interface
CCID	Chip Card Interface Device
CSS	Cascading Style Sheets
DOM	Document Object Model
ECC	Kryptografie eliptických křivek – Elliptic curve cryptography
GSAP	GreenSock Animation Platform
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IoT	Internet věcí – Internet of Things
IP	Internet Protocol
JSX	JavaScript XML
KVAC	Klíčově ověřitelná anonymní pověření – Keyed-Verification Anonymous Credentials
MAC	Message Authentication Code
MIT	Massachusetts Institute of Technology
NPAPI	Netscape Plugin Application Programming Interface
OS	Operační systém
RADIUS	Remote Authentication Dial In User Service
RKVAC	Klíčově ověřitelná anonymní pověření s revokací – Revocable Keyed-Verification Anonymous Credentials
TACACS	Terminal Access Controller Access-Control System
TCP	Transmission Control Protocol

USB Universal Serial Bus

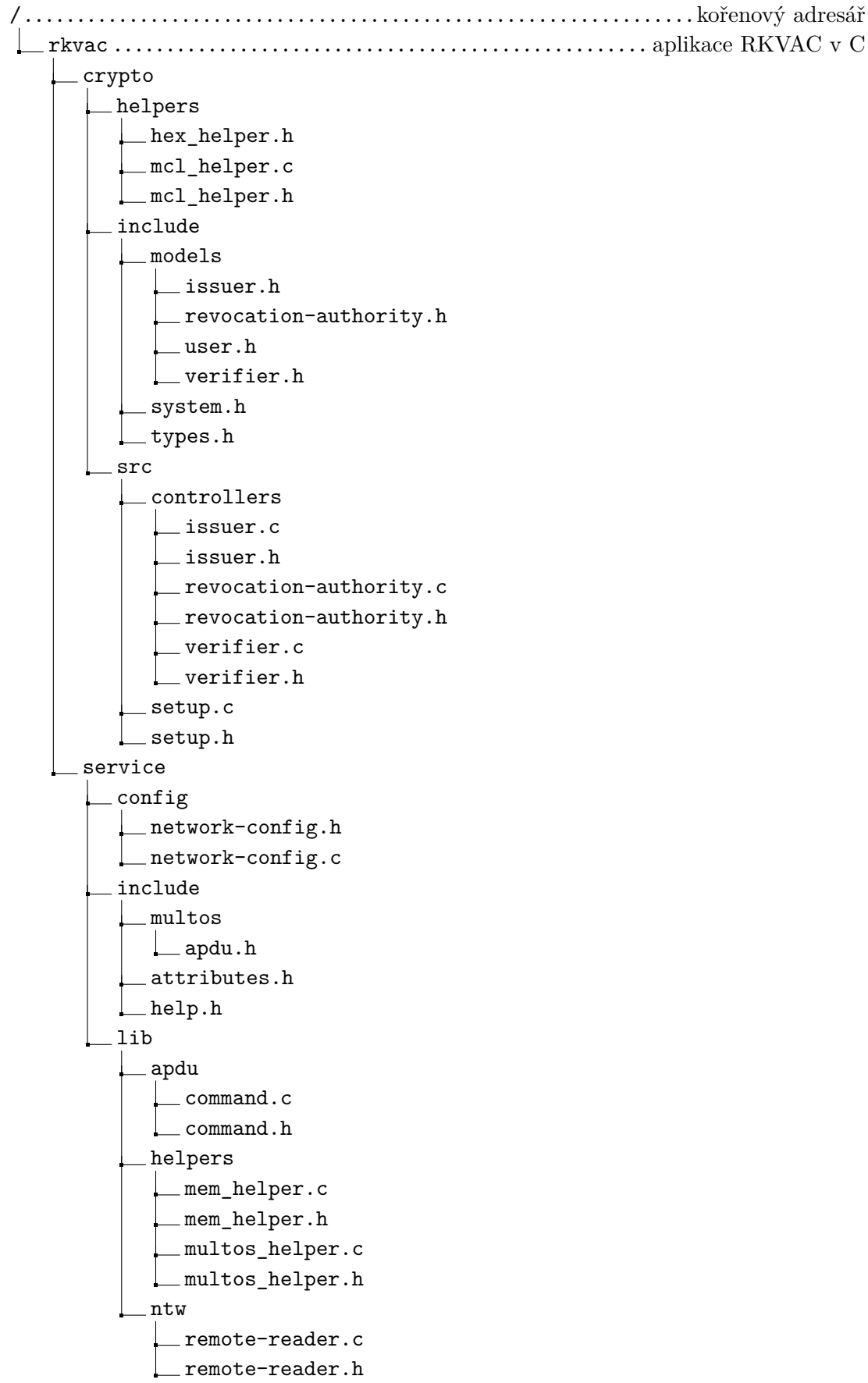
wBB weak Boneh-Boyen

A Seznam odevzdaných souborů

- / ..... kořenový adresář
  - client ..... klientská část
    - public
      - favicon.ico
      - index.html
      - logo192.png
      - logo512.png
      - manifest.json
      - robots.txt
    - src
      - api
        - api.js
        - connectCard.js
        - parseHelpers.js
      - assets
        - i18n
          - translations
            - cz.json
            - en.json
      - components
        - demo
          - DemoIssue.js
          - DemoRevoke.js
          - DemoSetup.js
          - DemoTest.js
          - DemoVerify.js
        - scheme
          - Canvas.js
        - Button.js
        - CardReader.js
        - CardReaders.js
        - Checkbox.js
        - Container.js
        - Credits.js
        - Icon.js
        - Input.js
        - InputRow.js
        - Layout.js
        - LoadingImg.js
        - MenuButton.js
        - ModalWindow.js
        - Notification.js
        - Radio.js
        - Tab.js

- / — kořenový adresář
  - client — klientská část
    - src
      - components
        - TabContent.js
        - Table.js
        - Tile.js
        - Wrappers.js
        - ZoomValue.js
      - helpers
        - helpers.js
        - themeConstants.js
      - images
        - entities
          - issuer.png
          - ra.png
          - user.png
          - verifier.png
        - scheme
          - attributes.png
          - attributes_en.png
          - attributes_show.png .6 attributes_show_en.png
          - credential.png
          - credential_en.png
          - credential_random.png
          - credential_random_en.png
          - entity.png
          - epoch.png
          - epoch_en.png
          - i_key.png
          - i_key_en.png
          - issue.png
          - issuer.png
          - nonce.png
          - prove.png
          - prove_en.png
          - pseudonym.png
          - ra.png
          - ra_handler.png
          - ra_handler_en.png
          - ra_key.png
          - ra_key_en.png
          - request.png
          - request_en.png
          - revoke.png
          - rh.png

```
/ ..... kořenový adresář
├── client ..... klientská část
│   ├── src
│   │   ├── images
│   │   │   ├── scheme
│   │   │   │   ├── rl.png
│   │   │   │   ├── user.png
│   │   │   │   ├── user_id.png
│   │   │   │   ├── verifier.png
│   │   │   │   ├── verify.png
│   │   │   │   ├── vi_key.png
│   │   │   │   └── vi_key_en.png
│   │   │   └── loading.gif
│   │   ├── pages
│   │   │   ├── Demo.js
│   │   │   ├── Description.js
│   │   │   ├── Entities.js
│   │   │   └── Protocols.js
│   │   ├── App.js
│   │   ├── App.test.js
│   │   ├── globalStyles.js
│   │   ├── i18n.js
│   │   ├── index.js
│   │   ├── setupTests.js
│   │   └── webcard.js
│   ├── .env
│   ├── .eslintrc.js
│   ├── .gitignore
│   ├── .prettierrc.js
│   ├── README.md
│   ├── package.json
│   └── package-lock.json
└── rkvac ..... aplikace RKVAC v C
    ├── cmake
    │   └── Modules
    │       ├── FindMCL.cmake
    │       └── FindPCSC.cmake
    ├── crypto
    │   ├── config
    │   │   └── config.h
    │   └── helpers
    │       ├── file_helper.c
    │       ├── file_helper.h
    │       ├── hash_helper.c
    │       ├── hash_helper.h
    │       └── hex_helper.c
```


/ kořenový adresář
rkvac aplikace RKVAC v C
crypto
helpers
hex_helper.h
mcl_helper.c
mcl_helper.h
include
models
issuer.h
revocation-authority.h
user.h
verifier.h
system.h
types.h
src
controllers
issuer.c
issuer.h
revocation-authority.c
revocation-authority.h
verifier.c
verifier.h
setup.c
setup.h
service
config
network-config.h
network-config.c
include
multos
apdu.h
attributes.h
help.h
lib
apdu
command.c
command.h
helpers
mem_helper.c
mem_helper.h
multos_helper.c
multos_helper.h
ntw
remote-reader.c
remote-reader.h

```
/ ..... kořenový adresář
├── rkvac ..... aplikace RKVAC v C
│   ├── service
│   │   ├── lib
│   │   │   └── pcsc
│   │   │       ├── reader.c
│   │   │       └── reader.h
│   │   └── src
│   │       └── controllers
│   │           ├── multos
│   │           │   ├── user.c
│   │           │   └── user.h
│   │           └── remote
│   │               ├── user.c
│   │               └── user.h
│   ├── CMakeLists.txt
│   ├── LICENSE.md
│   ├── README.md
│   └── main.c
├── server ..... serverová část
│   ├── .gitignore
│   ├── .prettierrc.js
│   ├── api.js
│   ├── package.json
│   ├── package-lock.json
│   ├── README.md
│   └── server.js
└── webcard ..... doplněk Webcard
    ├── extension
    │   ├── background.js
    │   ├── content.js
    │   ├── icon16.png
    │   ├── icon48.png
    │   ├── icon128.png
    │   └── manifest.json
    └── native
        ├── install.sh
        ├── json.hpp
        ├── makefile
        ├── org.cardid.webcard.native.json
        ├── uninstall.sh
        ├── webcard.cpp
        ├── webcard.h
        ├── webcard.ico
        ├── webcard.vcxproj
        └── webcard.wxs
```

```
/ ..... kořenový adresář
├── webcard ..... doplněk Webcard
│   ├── .gitignore
│   ├── LICENSE.md
│   └── README.md
├── .env
├── .gitignore
└── README.md ..... informace a pokyny ke spuštění
```