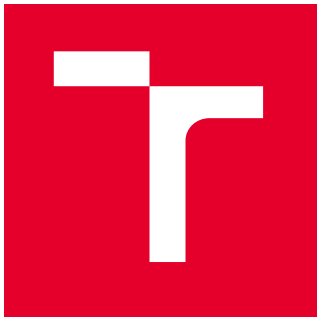


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ

DEPARTMENT OF TELECOMMUNICATIONS

PŘÍSTUPOVÝ SYSTÉM VYUŽÍVAJÍCÍ VÍCEÚROVŇOVOU AUTENTIZACI

ACCESS CONTROL SYSTEM USING MULTILEVEL AUTHENTICATION

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Tadeáš Cvrček

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Petr Dzurenda, Ph.D.

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Tadeáš Cvrček ***ID:*** 203699

Ročník: 2 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Přístupový systém využívající víceúrovňovou autentizaci

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s platformou čipových karet MultOS a vývojem aplikací pro chytrá zařízení s OS Android. Navrhněte přístupový systém využívající chytrá zařízení Android (telefon, hodinky) s podporou komunikačních technologií NFC a BLE. Systém bude podporovat víceúrovňovou autentizaci (multi-factor/multi-device authentication) a zajišťovat autentičnost, důvěrnost a integritu přenášených dat. Systém bude také implementovat funkční klíčový management. Platforma MultOS bude sloužit jako SAM modul pro přístupový terminál v podobě Raspberry Pi. Navrhnuté řešení implementujte, otestujte a proveďte výkonové testy.

DOPORUČENÁ LITERATURA:

[1] RANKL, Wolfgang. Smart Card Applications: Design Models for Using and Programming Smart Cards. 2007. 217 s. ISBN 9780470058824.

[2] DZURENDA, Petr; RICCI, Sara; CASANOVA MARQUÉS, Raul; HAJNÝ, Jan; ČÍKA, Petr. Secret Sharing-based Authenticated Key Agreement Protocol. In International Workshop on Security and Privacy in Intelligent Infrastructures (SP2I 2021) at the 16th International Conference on Availability (ARES 2021). 2021. p. 1-10. ISBN: 978-1-4503-9051-4.

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 24.5.2022

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

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

ABSTRAKT

Diplomová práce se věnuje protokolům s autentizovanou domluvou na společném klíči relace a implementaci těchto protokolů na čipových kartách MultOS a zařízeních s operačním systémem Google Android. Implementovaná uživatelská aplikace využívá komunikaci s mobilním telefonem přes NFC rozhraní, chytrými hodinkami pomocí technologie Bluetooth a čipovou kartu jako SAM (z angl. Secure Access Module) modul v roli ověřovatele. Kód je zároveň přenositelný a opakovaně využitelný v jiných projektech.

KLÍČOVÁ SLOVA

MultOS, Google Android, WearOS, Android SDK, Android NDK, SAM modul, čipová karta, autentizace, autentizační systémy, NFC rozhraní, Bluetooth technologie, biometrie

ABSTRACT

The master's thesis focuses on authenticated key agreement protocols and their implementation on MultOS smart cards platform and devices running Google Android operating system. Implemented applications use NFC (Near Field Communication) interface for communication with cellphone, Bluetooth technology for communication with smart watch and smart card as SAM (Secure Access Module) module in a verifier role. The source code is universal, so it is possible to use it in other projects.

KEYWORDS

MultOS, Google Android, WearOS, Android SDK, Android NDK, Secure Access Module, authentication, authentication systems, NFC interface, Bluetooth technology, biometrics

CVRČEK, Tadeáš. *Access control system using multilevel authentication*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2021, 57 s. Diplomová práce. Vedoucí práce: Ing. Petr Dzurenda, PhD.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Tadeáš Cvrček
VUT ID autora:	203699
Typ práce:	Diplomová práce
Akademický rok:	2021/22
Téma závěrečné práce:	Access control system using multilevel authentication

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

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

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

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

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Cílem bezpečných systémů je zajišťovat autentizaci, autentičnost, důvěrnost a integritu dat. S autentizací se lze setkat kdekoliv, kde dochází k prokazování identity člověka nebo i elektronických zařízení. V současné době rozrůstající se prostředí IoT (z angl. Internet of Things) zahrnuje integraci malých, nevýkonných, avšak účelných zařízení do společné sítě. Tímto je umožněno například připojit běžné domácí spotřebiče do sítě internet nebo sbírat data ze senzorů rozmístěných na různých místech po celém světě. Zahrnuta je však i průmyslová sféra, a to kvůli příchodu Průmyslu 4.0 [1]. Ten přichází s myšlenkou propojení průmyslových zařízení komunikační sítí a předávání dat. K tomu ale je potřeba určitého zabezpečení, protože jinak by hrozilo kompromitování přenosu útočníkem. To by znamenalo možné ohrožení aktiv subjektů, v krajních případech i subjektů samotných.

Implementace kryptografických schémat vyžaduje využívání kryptograficky bezpečných funkcí. K tomu je vhodný specializovaný hardware, například v podobě čipových karet. Některé čipové karty platformy MultOS umožňují hardwarovou akceleraci kryptografie, a mimo jiné mají také bezpečný paměťový systém. Z takových čipových karet je možné vyvinout vlastní SAM moduly (z angl. *Secure Access Module*), které následně slouží pro domluvu na společném klíči nebo jako úložiště citlivých informací. SAM moduly jsou obecně vzato integrované obvody, jako například ty v čipových kartách, které se zaměřují na zvýšení úrovně zabezpečení a podporují různá kryptografická primitiva [2].

Práce se zaměřuje na problematiku autentizačních systémů a popisuje protokoly, které jsou použity v rámci praktické implementace. Autentizace je z pohledu systémů velice důležitá a v době, kdy se rozrůstá sféra IoT se její důležitost ještě zvětšuje současně s rostoucími nároky na celkové zabezpečení systémů. Praktická část se věnuje popisu implementace konkrétních protokolů. Protokoly jsou cíleny na využití SAM modulů v podobě čipových karet k ověření uživatele, který do systému vstupuje prostřednictvím svého mobilního telefonu s operačním systémem Google Android. Uživatel je v rámci své mobilní aplikace autentizován pomocí znalosti kódu PIN nebo otisku prstu, pokud to mobilní telefon podporuje, a zároveň pomocí vlastních chytrých hodinek.

1 Přístupový systém

Systémy informačních technologií využívají různé mechanizmy k zajištění řízení přístupu. Řízení přístupu je postaveno na procesech autentizace, autorizace a jejich kombinaci [3]. S přístupovými systémy se lze nejčastěji setkat ve sdílených prostorách.

Během procesu autentizace dochází k ověření, že entita, která se nějak propaguje v systému, je skutečně tím, za koho se vydává [4]. Zjednodušeně lze říci, že jde o ověření identity. Následuje proces autorizace, jenž zjišťuje, jaká oprávnění v systému entita má. Mezi oprávnění může patřit například registrace jiných entit do systému, ale také jiné aktivity. Autentizovat entitu lze pomocí faktoru:

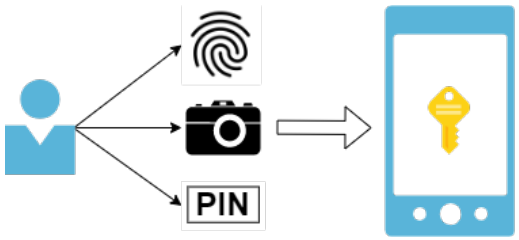
- **znalosti**: co entita zná, například znalost hesla, PIN kódu apod.,
- **vlastnictví**: co má entita ve svém vlastnictví, například že entita je držitelem specifické chytré karty, hardwarového klíče apod.,
- **jedinečných vlastností (biometrie)**: obecně čím entita je, například otisk prstu nebo skenování sítnice.

Proces autorizace spočívá v určení, zda entita má povolený přístup do systému nebo jeho části, popřípadě k jakým aktivitám [5]. Tyto pravomoci jsou delegovány administrátorem systému nebo entitou, kterou k tomu administrátor pověří. Reálně se lze s autorizací setkat například při práci se souborovými systémy při sdílení úložiště vícero uživateli.

1.1 Vícefaktorová autentizace

Při kombinaci různých faktorů vytvořit vícefaktorový autentizační systém [6]. V případě použití tohoto přístupu je možné systému nastavit, jaký je minimální počet faktorů, kterým se entita musí prokázat. Takový systém může entitu ověřovat například na základě znalosti hesla a vlastnictví hardwarového klíče. Pokud jsou oba faktory úspěšně ověřeny, je ověřen i uživatel. Mobilní telefony umožňují autentizaci pomocí více faktorů (otisk prstu, snímek obličeje, PIN kód apod.), jak je ukázáno v diagramu na obrázku 1.1, většinou však stačí, aby byl ověřen jen jeden faktor uživatele.

V některých případech se zavádí i legislativní požadavky na víceúrovňové zabezpečení. V roce 2018 přichází v platnost nařízení Evropské unie 2018/389 (známé také pod označením PSD2 SCA, z angl. *Payment Services Directive 2 Strong Customer Authentication*), které rozšiřuje směrnici 2015/2366 a popisuje autentizaci platebních příkazů [7]. Dle PSD2 SCA je při potvrzení platebního příkazu potřeba autentizovat uživatele aspoň dvoufázově (například kombinací hesla a aplikace banky v mobilním zařízení uživatele). Do české legislativy se toto nařízení dostává prostřednictvím



Obr. 1.1: Různé metody autentizace uživatele vůči mobilnímu telefonu.

§ 223 zákona 370/2017 Sb. o platebním styku [8].

1.2 Protokoly pro sjednání klíče a autentizaci

Kryptografická AKA (z angl. *Authentication and Key Agreement*) schémata se vyznačují sjednáním symetrického klíče mezi dvěma stranami, a současně autentizací stran vůči sobě [9]. Například lze tímto způsobem zkombinovat DH (z angl. *Diffie-Hellman*) protokol s DSA (z angl. *Digital Signature Algorithm*) schématem.

V současné době jsou AKA schémata aplikována například na poli mobilních komunikačních technologií, jako jsou 3G nebo 5G sítě. Konkrétně 5G sítě využívají mimo jiné protokol EAP-AKA (z angl. *Extensible Authentication Protocol - Authentication and Key Agreement*), který je popsán v dokumentu RFC 4187, jenž popisuje autentizaci a distribuci klíčů relací v mobilních sítích [10].

1.2.1 Asymetrický AKA protokol

Implementovaný asymetrický AKA protokol (zkr. *ASAKA protokol*) je založen na operacích s body na eliptických křivkách a je popsán v článku [11]. Pro vývoj byla zvolena křivka `secp256k1` a hašovací funkce `SHA-1`. Algoritmus je rozdělen na část ověřovatele (algoritmus 1), část uživatele (algoritmus 2) a část zařízení uživatele (algoritmus 3).

Protokol začíná algoritmem 1. Je vygenerována náhodná hodnota r_S ověřovatele. Ověřovatel dále získá bod t_S vynásobením generujícího bodu křivky hodnotou r_S, a tím vytvoří kryptografický závazek. Hodnotu e_S získá ověřovatel vytvořením haše z hodnot Y a t_S. Nakonec první části vypočítá hodnotu s_S a předá ji, společně s hodnotami e_S a Y, uživateli. Uživatel dle algoritmu 2 zrekonstruuje hodnotu t_S, kterou si uloží jako t'_S. Spočítá haš z hodnot Y a t'_S a zkontroluje, zda-li se výsledek shoduje s hodnotou e_S.

Algoritmus 1 `ASAKA-VERIFIER-SIGNVERIFY`(Y, sk_S, pk_C)

1: $r_S \in Z_q^*$
2: $t_S \leftarrow g^{r_S}$
3: $e_S \leftarrow \mathcal{H}(Y, t_S)$
4: $s_S \leftarrow r_S - e_S \cdot sk_s \bmod q$
5: **Send:** $Y, \sigma = (e_S, s_S)$ ⇒ `ASAKA-USER-PROOFVERIFY`
6: **Receive:** $\pi = (e_C, s_C)$ ⇐ `ASAKA-USER-PROOFVERIFY`
7: $t' \leftarrow g^{s_C} \cdot pk_C^{e_C}$
8: $\kappa \leftarrow t'^{r_S}$
9: $\tau_S \leftarrow e_C \stackrel{?}{=} \mathcal{H}(Y, t', \kappa)$
10: **return** $\tau_S = 0/1, \kappa$

Uživatel otevře komunikační kanál se svým zařízením (například chytrými hodinkami), na kterém je spuštěný algoritmus 3, a odešle mu zrekonstruovaný bod t'_S na eliptické křivce. Zařízení vygeneruje svoji hodnotu r_i, kterou si uloží do paměti po zbytek procesu autentizace. Hodnotu t_i spočítá jako skalární součin generujícího bodu eliptické křivky a hodnoty r_i. Hodnotu bodu t'_S vynásobí skalárem r_i a získá tak bod κ_i. Vypočítané souřadnice bodů t_i a κ_i vrátí uživateli. Komunikační kanál mezi uživatelem a zařízením zůstává otevřený.

Algoritmus 2 `ASAKA-CLIENT-PROOFVERIFY`(Y, σ, pk_S, sk_C)

1: $t'_S \leftarrow g^{s_S} \cdot pk_S^{e_S}$
2: $\tau_C \leftarrow e_S \stackrel{?}{=} \mathcal{H}(Y, t'_S)$
3: **if** $\tau_C = 0$ **then exit**
4: **Send:** t'_S ⇒ `ASAKA-DEVICE-PROOF`
5: **Receive:** κ_i, t_i ⇐ `ASAKA-DEVICE-PROOF`
6: $r_C \in_R \mathbf{Z}_q^*$
7: $t \leftarrow g^{r_0} \cdot \kappa_i$
8: $\kappa \leftarrow t'^{r_C}_S \cdot \kappa_i$
9: $e_C \leftarrow \mathcal{H}(Y, t, \kappa)$
10: **Send:** e_C ⇒ `ASAKA-DEVICE-PROOF`
11: **Receive:** s_i ⇐ `ASAKA-DEVICE-PROOF`
12: $s_{PIN} \leftarrow \mathcal{H}(PIN)$
13: $s_0 \leftarrow r_C - e_C \cdot (sk_C + s_{PIN}) \bmod q$
14: $s_C \leftarrow s_0 + s_i \bmod q$
15: **return** $\tau_C = 0/1, \pi = (e_C, s_C), \kappa$

Algoritmus 3 `ASAKA-DEVICE-PROOF`(t'_S, sk_i)

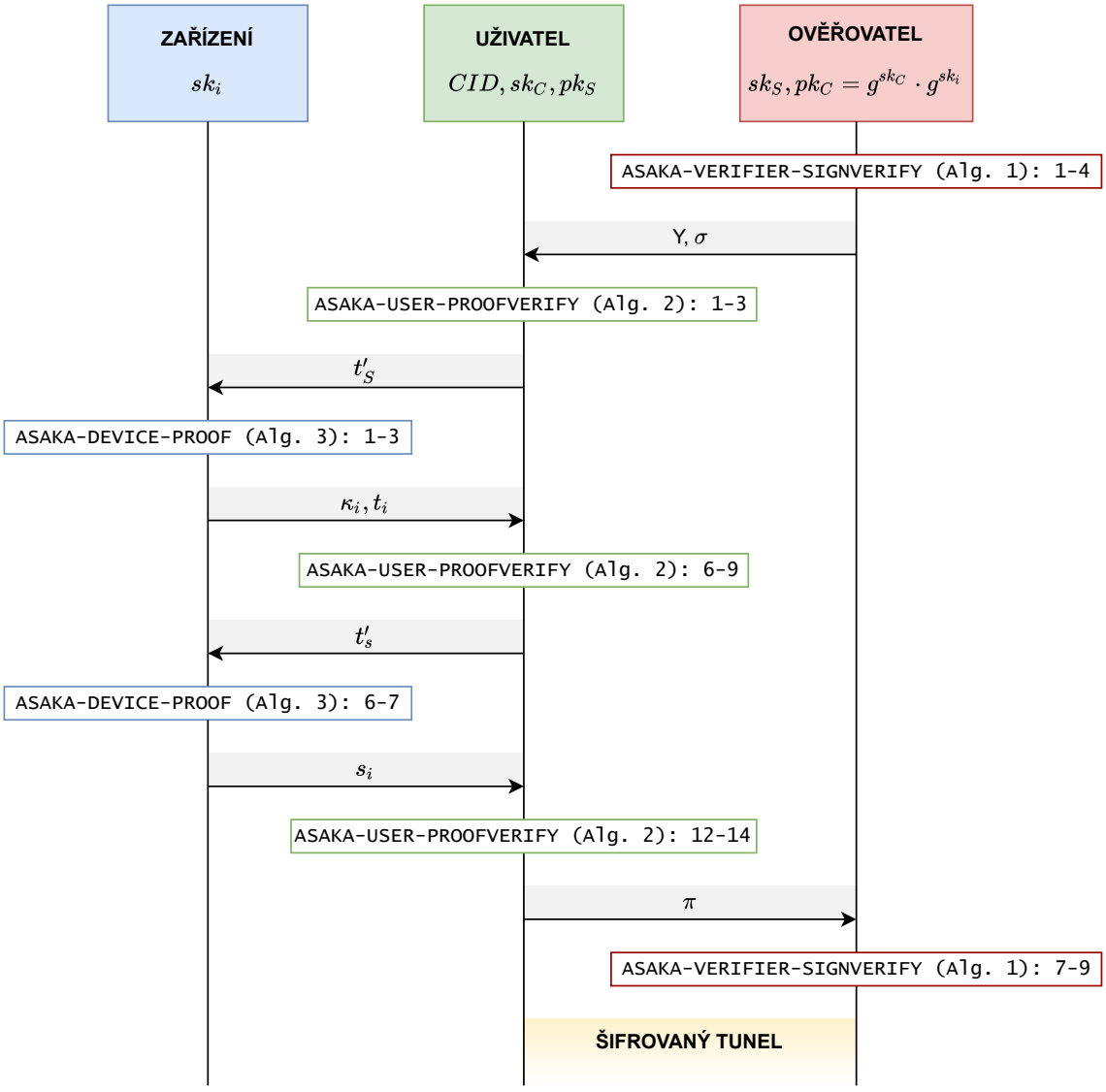
1: $r_i \in Z_q^*$
2: $t_i \leftarrow g^{r_i}$
3: $\kappa_i \leftarrow t_S'^{r_i}$
4: **Send:** κ_i, t_i $\Leftarrow$ `ASAKA-CLIENT-PROOFVERIFY`
5: **Receive:** e_C $\Rightarrow$ `ASAKA-CLIENT-PROOFVERIFY`
6: $s_i \leftarrow r_i - e_C \cdot sk_i \bmod q$
7: **Return:** s_i

Uživatel si dále dle algoritmu 2 vygeneruje vlastní náhodnou hodnotu r_C, kterou vynásobí generující bod křivky a hodnotu si uloží jako t. K bodu t zároveň uživatel přičte bod t_i získaný od jeho zařízení. Jako bod κ si uloží výsledek po vynásobení bodu t'_S skalárem r_C, ke kterému přičte bod κ_i zařízení, a tuto hodnotu pak použije, společně s t a Y, při vytvoření haše e_C.

Na uživatelovo zařízení je odeslána hodnota e_C. Zařízení díky tomu může spočítat důkaz znalosti s_i (algoritmus 3, řádky 6 až 7). Důkaz znalosti je odeslán uživateli a komunikace mezi zařízením a uživatelem končí.

Na závěr algoritmu 2, tj. řádky 12 až 14, spočítá uživatel hodnotu s_C, která je ovlivněna důkazem znalosti s_i a hodnotou s_PIN, která je derivovaná z PIN kódu uživatele. Ověřovateli pak zasílá informaci o výsledku kontroly haše, hodnoty e_C, s_C a κ. Ověřovatel dle řádků 7 až 9 algoritmu 1 zrekonstruuje hodnoty t' a κ. Zrekonstruované hodnoty a hodnotu Y nakonec předá hašovací funkci a porovná výstup s hodnotou e_C od uživatele. Hodnoty by se měly rovnat.

Grafické znázornění komunikace a jednotlivých kroků je na obrázku 1.2. Díky diagramu lze lépe vidět a pochopit komunikaci mezi jednotlivými entitami podílejícími se na protokolu. Celkem se jedná tedy ověřovatele, uživatele a uživatelovo zařízení. Je vidět, že během vykonávání protokolu je provedeno celkem šest transakcí dat, z toho dvě mezi uživatelem a ověřovatelem. Výstupem protokolu je autentizace entit a symetrický klíč pro vytvoření zabezpečeného šifrovaného tunelu, jehož lze využít pro následný přenos dat.



Obr. 1.2: Diagram ASAKA protokolu.

2 IoT prostředí

Jelikož se v IoT využívají nepříliš výkonná zařízení, při návrhu zabezpečeného systému je třeba počítat s určitými omezeními. Především se může jednat o nedostatečnou paměťovou kapacitu, nízký výpočetní výkon nebo naprostou absenci kryptografického koprocesoru, který by dopomohl hardwarovou implementací k výpočtům kryptografických primitiv.

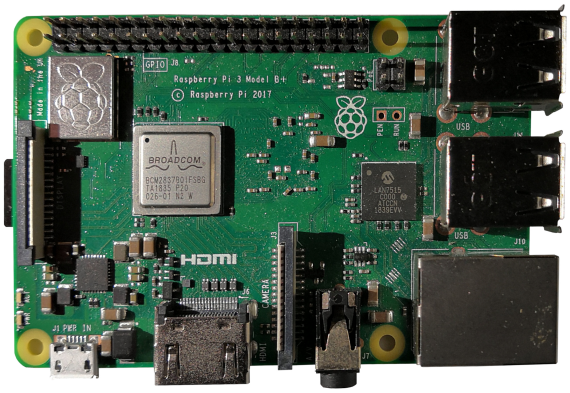
Pro vývoj systémů na bázi mikrokontrolérů, což jsou čipová řešení zahrnující paměť, procesor nebo i řadiče, je velice populární platforma Arduino [12] [13]. Platforma nabízí mnoho modelů, se kterými může vývojář pracovat. Asi nejpopulárnější je model UNO, jenž je postavený na osmibitové architektuře AVR a je relativně levný. Pokud je potřeba rozměrově menší zařízení, je k dispozici výkonově srovnatelný model NANO. Pokud je do systému nutné implementovat náročnější program, pak platforma Arduino nabízí také model DUO, postavený na architektuře ARM (z angl. *Advaced RISC Machines*). Žádný z modelů však z výroby nedisponuje kryptografickým koprocesorem, který by bylo možné přímo využít, proto řešením je pomalejší softwarová implementace [14]. Celkově jsou tedy kryptografická primitiva, jako například modulární aritmetika nebo operace na eliptických křivkách, na této platformě pomalejší. Pokud je vyžadován vyšší výkon, je k dispozici platforma Raspberry Pi, které je věnovaná část Platforma Raspberry Pi.

2.1 Platforma Raspberry Pi

Raspberry Pi je značka jednodeskových počítačů malých rozměrů od britské společnosti Raspberry Pi Foundation [15]. Původním účelem těchto počítačů bylo podpořit výuku programování a elektrotechniky, a současně zachovat přijatelnou cenu těchto zařízení. První model byl vydán v roce 2012.

Celá platforma je založena na procesorech architektury ARM. Výbava se liší v závislosti na generaci a konkrétním modelu. Téměř každé zařízení má GPIO (z angl. *General-Purpose Input/Output*) konektor, sloužící k připojení periferii nebo k ovládání obvodů vlastní výroby. Ve standardní výbavě jsou také konektory HDMI (z angl. *High-Definition Multimedia Interface*), USB (z angl. *Universal Serial Bus*) a RJ-45. To se ale netýká modelu Raspberry Pi Pico, který se řadí mezi mikrokontroléry a má pouze GPIO konektor, a také speciálních výpočetních modulů [16]. Prvním modelem, který byl vybaven 64-bitovým procesorem, bylo Raspberry Pi 3 B (na obrázku 2.1). V roce 2020 byl vydán model Raspberry Pi 400 integrovaný přímo do klávesnice.

V současné době (rok 2021) je aktuální již čtvrtá generace zařízení [17]. Tato generace je založena na čtyřjádrovém procesoru Broadcom BCM2711 s taktem 1,5



Obr. 2.1: Raspberry Pi 3. generace (model B+).

GHz. K dispozici jsou verze s 2, 4 nebo 8 GB operační paměti. Úložiště je zprostředkováno microSD kartou. Síťové připojení je možné buď přes Wi-Fi (standard IEEE 802.11ac) nebo přes gigabitovou síťovou kartu s konektorem RJ-45. Komunikovat s okolními zařízeními lze pomocí technologií Bluetooth (verze 5.0) a BLE.

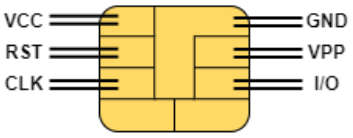
Kromě standardních modelů existují také modely, které jsou navrženy pro konkrétní využití. Tyto modely často neobsahují komponenty, které pro svůj účel nepotřebují, a proto jsou také levnější. V roce 2014 byl vydán první tzv. Výpočetní modul. Jedná se v podstatě o Raspberry Pi bez klasických konektorů a s deaktivovanou grafickou jednotkou. Modul využívá DDR2 SO-DIMM konektor, kterým se připojuje k hlavní desce. Deska může mít připojeno více modulů. Pro sféru IoT byl také vytvořen model Zero. Vydán poprvé v roce 2015 obsahoval pouze 512 MB operační paměti a jednojádrový procesor s taktem 1 GHz (ve verzi W s podporou Wi-Fi a Bluetooth). V roce 2021 byla potvrzena druhá generace s čtyřjádrovým procesorem. Vývoj systémů s využitím mikrokontrolérů je možný pomocí Raspberry Pi Pico. Jedná se o desku s GPIO konektivitou a čipem RP2040, který byl navržen konkrétně pro tento model.

2.2 Čipové karty

Čipová karta je fyzické zařízení s vestavěným integrovaným obvodem a rozhraním [18]. Integrovaným obvodem může být paměťový čip nebo procesor, případně může karta obsahovat i kryptografický koprocesor. Komunikaci s kartou je možné provádět přes kontaktní rozhraní, jak je definováno normami ISO/IEC 7810 a ISO/IEC 1816, nebo bezkontaktně, jak je definováno normou ISO/IEC 14443. Kontaktní rozhraní je vykresleno na obrázku 2.2 a jednotlivé piny označují:

1. **VCC**: napájení,
2. **RST**: reset,

3. **CLK**: takt (synchronizace komunikace),
4. **GND**: uzemnění,
5. **VPP**: programovací pin,
6. **I/O**: sériové rozhraní (poloviční duplex).



Obr. 2.2: Rozhraní a zapojení kontaktní čipové karty

Integraci komunikace čipových karet s počítači zajišťuje specifikace PC/SC (z angl. *Personal Computer/Smart Card*). Implementace je přímo dostupná v rámci operačního systému Microsoft Windows a macOS [19]. Pro použití na operačním systému Linux je potřeba nainstalovat příslušné balíčky. Pro Debian Linux a jeho deriváty je to například balíček `pcscd`.

Tab. 2.1: Podpora kryptografických primitiv na čipových kartách.

	Java Card	**Basic Card**	**MultOS**	**.NET Card**
DES	ANO	ANO	ANO	ANO
3DES	ANO	ANO	ANO	ANO
AES	ANO	ANO	ANO	ANO
RSA	ANO	ANO	ANO	ANO
DSA	ANO	NE	NE	NE
ECDSA	ANO	ANO	ANO	NE
ECDH	ANO	ANO	ANO	NE
SHA-1	ANO	ANO	ANO	ANO
SHA-2	ANO	ANO	ANO	ANO
SHA-3	ANO	NE	NE	NE
MD5	ANO	NE	NE	ANO
MOD[1]	NE	ANO	ANO	NE
ECOP[2]	NE	ANO	ANO	NE

[1]Modulární aritmetika

[2]Operace na eliptických křivkách

Existují různé typy čipových karet. Jednak je tu platforma Java Card, která umožňuje programování pomocí jazyka Java. Dále pak existuje platformy Basic Card s podporou programování v jazyce Basic, .NET Smart Card a v neposlední řadě, v této práci využitá, platforma MultOS [20][21]. Všechny zmíněné produkty se liší

hardwarem, a proto také hardwarovou podporou akcelerace kryptografických primitiv, jak je ukázáno v tabulce 2.1 [22]. Před implementací kryptografických schémat je vhodné se prvně podívat, jestli je vybraný typ karty vhodný. Případná kombinace hardwarové akcelerace se softwarovou implementací některých funkcí může způsobit negativní dopad na výkon a čas potřebný ke zpracování.

2.2.1 Komunikační protokoly

Pro komunikaci mezi čtečkou a čipovou kartou se používají APDU (z angl. *Application Protocol Data Unit*) zprávy [23]. Jejich struktura je definovaná normou ISO/IEC 7816-4, která byla od vydaná v roce 1995, pak třikrát upravena, a to v letech 2005, 2013 a 2020 [24][25].

Pro přenos APDU zpráv byly definovány dva transportní protokoly, které se označují jako T0 a T1. Liší se maximálním množství přenositelných dat. Zatímco T0 umožňuje přenést nanejvýš 255 B, T1 může až 65535 B.

APDU požadavek						
CLA	**INS**	**P1**	**P2**	**LC**	**DATA**	**LE**
1 B	1 B	1 B	1 B	0/1/3 B	0-255 B (T0)/0-65535 B (T1)	0/1/3 B

APDU odpověď		
DATA	**SW1**	**SW2**
0-255 B (T0)/0-65535 B (T1)	1 B	1 B

Obr. 2.3: Struktura APDU zpráv

APDU požadavek, který je znázorněn na obrázku 2.3 nahoře, začíná polem, o délce 1 B, označujícím třídu instrukce, která bude volána. Jakou instrukci má karta vykonat je v dalším poli, rovněž o délce 1 B. Existují specializované instrukce, které mají předdefinované identifikátory (například instrukce `GET RESPONSE` má přidělený identifikátor `0xC0`). Další 2 B jsou rezervované pro dva parametry, které jsou předány pro zpracování v rámci implementované instrukce na čipové kartě. Kolik dat je zasíláno je vyjádřeno v následujícím poli, a to o délce 0, 1 nebo 3 B (záleží na zvoleném transportním protokolu a na množství přenášených dat). Následuje datové pole o velikosti dané transportním protokolem, a to 0 až 255 B, resp. 65535 B. Poslední pole APDU požadavku má délku 0, 1 nebo 3 B a vyjadřuje očekávanou délku návratových dat z čipové karty.

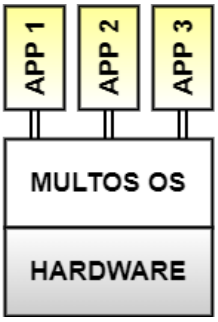
Čipová karta odpovídá na APDU požadavek odpovědí, jež je znázorněna na obrázku 2.3 dole. Začíná datovým polem o délce dané transportním protokolem, a to 0 až 255 B, resp. 65535 B. Toto pole ze své definice není povinné. Povinné jsou ale pole

označující výstupní stav vykonané instrukce a pro jejich hodnoty jsou rezervovány vždy poslední 2 B APDU odpovědi.

Princip APDU zpráv lze využít i jinak, než pouze pro komunikaci s čipovými kartami. Příkladem může být přenos dat mezi NFC (z angl. *Near Field Communication*) čtečkou a mobilním telefonem.

2.2.2 MultOS čipové karty

Technologie platformy MultOS se zaměřují na chytré karty s jistou úrovní zabezpečení [26]. Na čipových kartách je k dispozici proprietární operační systém typu RTOS (z angl. *Real-Time Operating System*), který zpřístupňuje API pro přímou komunikaci s hardwarovými prostředky karty, jak je zobrazeno na obrázku 2.4 [27]. Obrázek 2.5 pak ukazuje, jak fyzicky vypadá čipová karta společnosti MultOS.



Obr. 2.4: Schéma komunikace apletů s hardware MultOS.

Každá čipová karta MultOS má jinou výbavu. Především se jedná o rozdílné schopnosti, co se hardwarové akcelerace kryptografických algoritmů týče. Některé karty podporují DES či AES, ale nepodporují operace na eliptických křivkách nebo jiné kryptografické algoritmy.

Vývoj programů, které jsou někdy označovány také jako applety, probíhá hlavně v programovacím jazyce C. Případně lze programovat přímo v jazyce symbolických adres (angl. *assembly language*) nebo je možné využít i jazyk Java. Každý program má svůj unikátní identifikátor. V rámci programu jsou pak jednotlivé procesy rozděleny na takzvané instrukce, které lze volat pomocí APDU zpráv. Během komunikace s kartou lze také oběma směry předávat data.

Kromě specifické konfigurace hlavního procesoru a koprocesoru, což je nutné brát v potaz při výběru karet, mají karty také odlišné kapacity paměti. Paměť pro program je rozdělena na tři části:

- **globální paměť**: paměť, ke které lze přistupovat globálně (primárně je využívaná pro data APDU zpráv),

Obr. 2.5: Čipová karta MultOS.

- **paměť relace**: paměť, která se při vytvoření nové relace resetuje do výchozího stavu daného kódem,
- **statická paměť**: paměť určená hlavně pro statické hodnoty (například doménové parametry eliptických křivek), ale je možné ji přepisovat i za běhu programu.

Dále lze při výběru karet MultOS jako referenci použít oficiální dokumentaci, ve které je výpis podporovaných algoritmů a funkcí pro jednotlivé typy karet. Výběr karet lze pak zohlednit i podle toho. Během implementace protokolu ASAKA byla použita karta s označením `ML4-G21`. Odlišnosti mezi různými modely, co se podpory kryptografických funkcí týče, je možné potvrdit srovnáním s modelem `ML4-P17` v tabulce 2.2. Některé algoritmy je nutné volat přes speciální funkce. Například algoritmus 3DES v módu ECB je zde přítomen prostřednictvím volání instrukce `Block Encipher`, resp. `Block Decipher`, s parametrem `0xD8`, resp. `0xD9`.

Tab. 2.2: Podpora kryptografických funkcí na vybraných kartách MultOS.

Funkcionalita	**ML4-P17**	**ML4-G21**
3DES ECB	ANO	ANO
AES ECB	ANO[1]	NE
SHA-1	ANO	ANO
Modulární aritmetika	ANO[2]	ANO
Operace na eliptických křivkách	NE	ANO

[1]Voláno prostřednictvím instrukce `Block Encipher`, resp. `Block Decipher`

[2]Bez podpory modulární inverze

Výpis 2.1: Ukázka struktury paměti appletu MultOS v kódu.

```
/// Global values - RAM (Public memory)
#pragma melpublic
uint8_t apdu_data[APDU_L_MAX];

/// Session values - RAM (Dynamic memory)
#pragma melsession
uint8_t ve_nonce[NONCE_LENGTH];

/// Static values - EEPROM (Static memory)
#pragma melstatic
char *ve_string = "00000000Verifier";
```

Čipové karty nepodporují dynamickou alokaci paměti během běhu programu, a proto musí být veškerá paměť rozdělena již v momentě kompilace kódu, což je v kódu zapsáno jako na ukázce ve výpisu 2.1. Z toho vyplývá, že nedochází ani k uvolňování paměti za běhu. Již při návrhu programu je potřeba brát v potaz, jaké typy paměti budou konkrétně využity, a to z důvodu výkonnostních i bezpečnostních. Programy na kartě mohou zasahovat pouze do dat v paměťovém prostoru, který jim náleží. To zajišťuje integrovaný firewall.

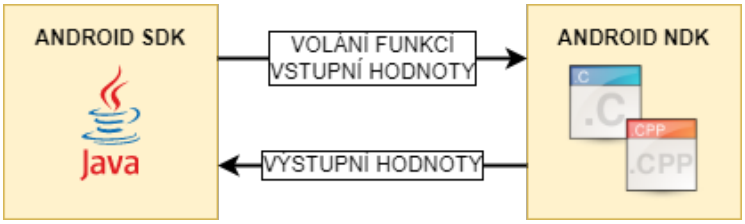
Pro kompilaci a nahrávání programů je k dispozici MultOS SDK (z angl. *Software Development Kit*) nástrojová sada. Při nahrávání do paměti čipové karty je přímo alokována struktura paměti dle nastavení.

2.3 Vývoj aplikací pro operační systém Google Android

Pro mobilní telefony a chytré hodinky s operačním systémem Google Android lze vyvíjet software prostřednictvím programu Android Studio, který je založený na řešeních od české firmy JetBrains [28]. Jedná se o oficiální způsob vývoje, který probíhá primárně pomocí programovacích jazyků Java a Kotlin [29]. Kotlin je nový programovací jazyk, který přímo z Javy vychází a je s ní plně kompatibilní. Stejně jako Java je Kotlin jazyk interpretovaný, což znamená, že při kompilaci dochází k překladu jen do bajtového kódu. Ten je do strojového sestaven až při spuštění na konkrétním zařízení nebo dokonce až v momentě, kdy má dojít ke spuštění konkrétní části aplikace, tzv. JIT kompilátor (z angl. *Just-In-Time*) [30]. V případě návrhu uživatelského rozhraní aplikací se využívá tagovací syntaxe XML. K sestavení výsledného balíčků slouží Android SDK.

Od verze Android Studio 2.2 je k dispozici také nástrojová sada Android NDK (z angl. *Native Development Kit*) [31]. Cílem této sady je umožnit vytvářet kód v jazycích C a C++. Kód programu je následně kompilován přímo do strojového kódu v závislosti na cílové architektuře, a to již při sestavování balíčku. Sestavování probíhá pomocí nástrojů CMake a Make. Výhodou tohoto přístupu je několik:

- **rychlejší zpracování dat**: instrukce zkompilované předem (a v lepším případě optimalizované již při vývoji) jsou procesorem vykonávány rychleji,
- **lepší možnosti optimalizace**: programátor může například lépe určovat způsoby manipulace s pamětí,
- **veřejně dostupný obsah**: využití knihoven napsaných v jazyce C nebo C++.



Obr. 2.6: Způsob práce s Android SDK a NDK současně.

Vývoj aplikace, která z části je psaná v jazyce C nebo C++, spočívá ve vytvoření dvou balíčků: balíček standardní aplikace na bázi Android SDK a balíček nativní aplikace Android NDK. Aplikace založená na Android NDK obsahuje funkce, které lze volat a vykonávají části programu implementované v jazyce C nebo C++. Této aplikaci lze ze standardní aplikace předávat data, a po vykonání funkcí číst návratové hodnoty. Kompatibilitu v tomto případě zajišťuje JNI (z angl. *Java Native Interface*) [32]. Diagram komunikace mezi standardní aplikací a nativní aplikací je vidět na obrázku 2.6. Kombinace standardní a nativní aplikace je vhodná v případě existence časově kritických částí, jako je tomu například u kryptografických primitiv [33].

Pomocí stejných nástrojů je možné vyvíjet také aplikace pro systém Wear OS, který je optimalizovaný pro nositelnou elektroniku, jako jsou například chytré hodinky, a přímo vychází z operačního systému Android. Díky tomu mohou aplikace sdílet stejný kód a implementovat stejná volání systémových funkcí.

2.4 Rozhraní Bluetooth

Komunikační rozhraní Bluetooth, které bylo poprvé popsané v roce 2002 v rámci IEEE 802.15, umožňuje vytvoření WPAN sítě (z angl. Wireless Personal Area Network) pro komunikaci mezi různými zařízeními [34]. Rozhraní bylo navrženo jako alternativa k fyzickému spojení pomocí drátu, současně také způsobem vhodným

pro využití v přenosných systémech. Bezdrátová komunikace využívá frekvence od 2.402 GHz do 2.480 GHz. Systém umí předcházet interferencím s okolními signály pomocí technologie AFH (z angl. Adaptive Frequency Hopping).

Spotřeba energie se odvíjí od použité třídy (angl. ozn. *CLASS*), která zároveň ovlivňuje i možný dosah signálu [35]. Nejčastěji je využívána implementace s výkonem dle specifikace CLASS 2. Vhodným výběrem třídy lze ovlivnit celkovou energetickou náročnost přenosu dat, což lze zohlednit dle potřebné vzdálenosti koncových bodů komunikace. Parametry jednotlivých tříd jsou vypsané v tabulce 2.3.

Tab. 2.3: Přehled specifikací tříd rozhraní Bluetooth.

Třída	CLASS 1	CLASS 2	CLASS 3
Max. výkon	100 mW	2,5 mW	1 mW
Max. přibližný dosah	100 m	10 m	1 m

Od vzniku tohoto standardu vzniklo několik verzí a vývoj stále pokračuje dál [36]. Historicky první standardizovaná byla až verze 1.1 schopná přenášet data rychlostí až 0.7 Mbit/s. Verze rozhraní 2.0 pak přinesla změnu kódování signálu a s tím navýšení maximální přenosové rychlosti na 3 Mbit/s, nižší spotřebu energie a lepší řešení interferencí s ostatními signály. Možnost spojení s přenosem dat prostřednictvím Wi-Fi rozhraní začala být podporována v roce 2009 od verze 3.0. Jedná se o volitelný způsob přenosu, který umožňuje využití standardu Bluetooth pro navázání spojení mezi zařízeními, která pak komunikují prostřednictvím Wi-Fi rychlostí až 24 Mbit/s. S verzí 4.0 bylo uvedeno rozhraní Bluetooth Low Energy pro chytrá zařízení s omezenými zdroji energie, jež bylo vylepšeno ve verzích 4.1 a 4.2 [37]. Bluetooth 5.0 rozšířil teoretický možný dosah signálu v otevřeném prostoru (v ideálním případě až na 200 m) a snížil celkovou spotřebu energie.

2.5 Rozhraní Near Field Communication

Bezdrátové komunikační rozhraní NFC nabízí zprostředkování komunikace mezi dvěma zařízeními v blízké vzdálenosti. Tato technologie, standardizovaná v rámci ISO/IEC 18092, je založena na standardu ISO/IEC 14443 a rozhraní FeliCa od společnosti Sony [38] [39]. Rozhraním je vybavena většina moderních mobilních telefonů.

Bezdrátový přenos probíhá na frekvenci 13,56 MHz a nabízí přenosovou rychlost od 106 kbit/s do 424 kbit/s, a to do vzdálenosti do 4 cm [40]. Technologicky musí být kompatibilní zařízení vybaveno magnetickou smyčkovou anténou a řídícím čipem, který je zpřístupněn ostatním komponentám zařízení. Zabezpečení přenosu

dat není řešeno na fyzické vrstvě, ale musí být implementováno na vyšších vrstvách komunikace.

NFC je často využíváno v platebních systémech a lze jej využít i k přenosu malého množství libovolných dat. Umožňuje také emulaci chytrých karet, což umožňuje emulovat implementaci čipové karty na NFC kompatibilním zařízení. Toho je využito i v rámci této diplomové práce.

3 Implementace přístupového systému

Implementace přístupového systému zahrnuje přípravu prostředí, instalaci potřebných nástrojů a nastavení sítě. Vývoj je prováděn vzdáleně, a to pomocí SSH (z angl. *Secure Shell*).

Celá struktura řešení je k dispozici v příloze B, a to včetně souboru `README.pdf`, který popisuje jednotlivé kroky ke spuštění. Zdrojový kód protokolu lze obecně rozdělit do částí:

- `asaka-security-scheme`: aplikace pro běh na zařízení Raspberry Pi, tj. ověřovacím terminálu, ke kterému je připojena alespoň jedna čtečka čipových karet a čtečka NFC,
- `pcsc-list`: aplikace pro načtení a zobrazení seznamu připojených zařízení kompatibilních PC/SC,
- `asaka-MultOS-verifier`: aplikace pro čipové karty MultOS připravená pro roli ověřovatele v autentizačním schématu (při připojení alespoň dvou čteček k zařízení),
- `AuthApp`: klientská aplikace pro mobilní telefony a chytré hodinky s operačním systémem Google Android, resp. Wear OS.

Pro implementaci protokolu byla použita upravená verze knihovny `micro-ecc`, která podporuje základní operace pro práci s body na eliptických křivkách. Podporu hašovacího algoritmu pak zajišťuje na klientském zařízení volně dostupná implementace `SHA1` [14] [41]. Obě zmíněné knihovny jsou naprogramované v jazyce C. Na mobilním telefonu bylo využito nástrojů Android NDK, které umožňují programování aplikací právě v tomto jazyce, popřípadě taktéž v jazyce C++.

3.1 Příprava prostředí

Běhové prostředí je založeno na operačním systému Raspberry Pi OS (dříve Raspbian), který je odvozený od linuxové distribuce Debian. z oficiálních webových stránek produktu lze nainstalovat různé varianty tohoto operačního systému, nicméně pro účely implementace protokolů je dostačující nainstalovat verzi s označením Lite, jež obsahuje pouze základní balíčky a například nemá přímou podporu grafického rozhraní, což lze případně vyřešit pozdější manuální instalací. Taková verze je rychlejší a úspornější na paměť. Dodatečně je potřeba doinstalovat následující balíčky systému `apt`:

- `zlib1g-dev`: obshuje kompresní funkce využívané balíčkem `libssl-dev`,
- `libssl-dev`: obsahuje kryptografické funkce,
- `libpcsclite-dev`: podpora PC/SC rozhraní pro komunikaci při vývoji,
- `cmake` a `make`: nástroje pro sestavení C/C++ aplikací,

- `pcscd`: služba, která umožňuje komunikaci PC/SC (musí být spuštěná).

Pro instalaci všech balíčků je možné použít instalační příkaz `apt-get install -y zlib1g-dev libssl-dev libpcsclite-dev make cmake pcscd`, který je však nutné spustit s příslušnými uživatelskými oprávněními pro instalaci balíčků do operačního systému. Parametr `-y` nastavuje všechny dotazy systému během instalace na hodnotu Ano (angl. *Yes*).

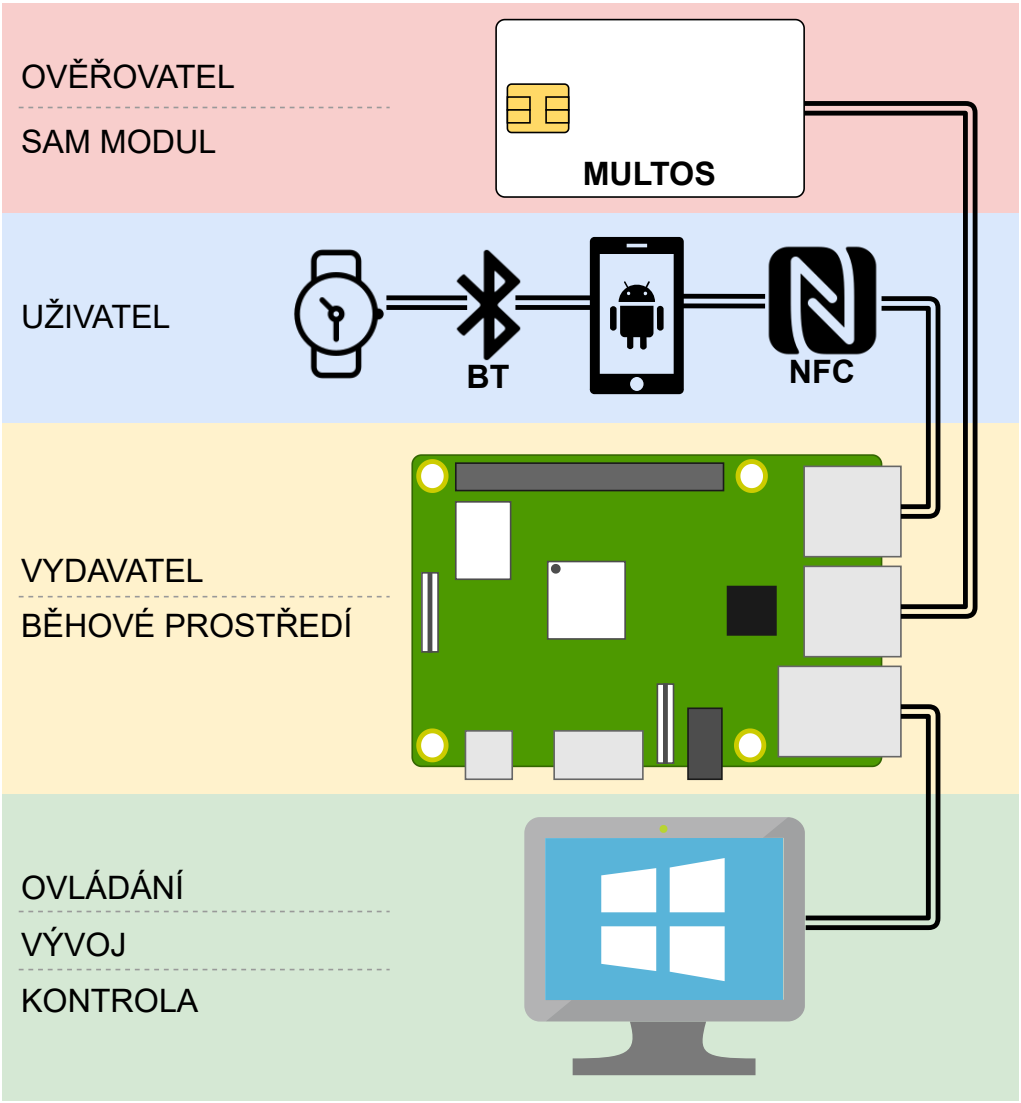
Vývoj aplikací pro čipové karty MultOS probíhá na počítači s operačním systémem Microsoft Windows. Pro tento operační systém totiž existuje nativní nástroj pro sestavení a nahrávání aplikací na čipové karty. Nahrávání obnáší analýzu paměťových nároků aplikace (pomocí nástroje `hls`) a samotné nahrání nástroji MultOS Smart Deck.

Jako univerzální vývojové prostředí (dále jen IDE, z angl. *Integrated Development Environment*) lze využít kupříkladu multiplatformní Microsoft Visual Studio Code, jehož obrazovka je zachycena v příloze A. Toto IDE umí pomocí rozšíření, která jsou k dispozici přímo z hlavního okna aplikace, zpřístupnit vývoj na vzdáleném zařízení pomocí protokolu SSH. Úpravy zdrojových kódů i sestavení v takovém případě probíhají na připojeném zařízení. Pomocí integrovaného terminálu lze zařízení také ovládat bez nutnosti instalace dalších komunikačních aplikací (například PuTTY). Tímto způsobem je vhodné vyvíjet pouze aplikace, které budou ve výsledném sestavení spouštěné právě na tomto zařízení.

Speciálně pro vývoj aplikací pro mobilní telefony s operačním systémem Android je třeba nainstalovat vývojové prostředí Android Studio, a to včetně nástrojů Android SDK. Využít lze opět operační systém Microsoft Windows, ale také jakoukoliv podporovanou linuxovou distribuci.

3.2 Popis přístupového systému

Schéma zapojení, včetně popisu účelů jednotlivých komponent, lze vidět na obrázku 3.1. Systém se skládá z ověřovatele reprezentovaného SAM modulem a připojeného k ověřovacímu terminálu přes čtečku karet. Uživatel vstupuje do systému pomocí rozhraní NFC, přes které může systém komunikovat s jeho mobilním telefonem a potažmo uživatelskou aplikací. Pomocí technologie Bluetooth může uživatelská aplikace mobilního telefonu komunikovat s chytrými hodinkami. Raspberry Pi v roli vydavatele poskytuje běhové prostředí terminálové aplikace a propojuje všechna rozhraní pomocí sběrnice USB. Pro ovládání, vývoj software a kontrolu logů lze využít počítač připojený přes síťové rozhraní. Ke komunikaci mezi Raspberry Pi a řídícím počítačem lze využít protokol SSH.



Obr. 3.1: Schéma zapojení systému.

Uživatel vstupuje do systému s chytrými hodinkami a mobilním telefonem podporujícím NFC a Bluetooth. V hodinkách i telefonu má předinstalovanou aplikaci implementující uživatelskou část protokolu, která je popsaná v části Asymetrický AKA protokol. Když aplikaci spustí poprvé, nebo v případě, že data aplikace byla resetována, musí provést proces personalizace během registrace v aplikaci. Uživatel je během procesu vyzván k zadání autentizačního faktoru, jako je například PIN kód, a je mu vygenerován pár klíčů. Během procesu personalizace mobilního telefonu dochází i k personalizaci chytrých hodinek. Pomocí nastavení může uživatel měnit typ autentizačního faktoru, zvolit autentizaci pomocí otisku prstu atd. Po úspěšně personalizaci je aplikace připravena provádět celou uživatelskou část protokolu, ale vždy až po autentizaci uživatele pomocí vybraného faktoru. Při procesu autentizace je telefon spojen se spárovanými hodinkami, jakožto dalším autentizačním faktorem,

pomocí rozhraní Bluetooth a se zbytkem systému pomocí rozhraní NFC. Výsledkem autentizace a domluvy na symetrickém klíči je vytvoření šifrovaného přenosového kanálu. Přes něj je z telefonu odeslán zašifrovaný časový token, který při úspěšném dešifrování ověřovatelem potvrzuje, že byl uživatel autentizován a má přístup do systému. Časový token má délku 11 B a jeho struktura je na obrázku 3.2. Obsahuje kontrolní vzestupnou sérii hodnot a zakódované aktuální kalendářní datum.

AUTENTIZAČNÍ ČASOVÝ TOKEN										
Kontrolní řada hodnot							Den	Měsíc	Rok	
00	01	02	03	04	05	06	DD	MM	YY	YY

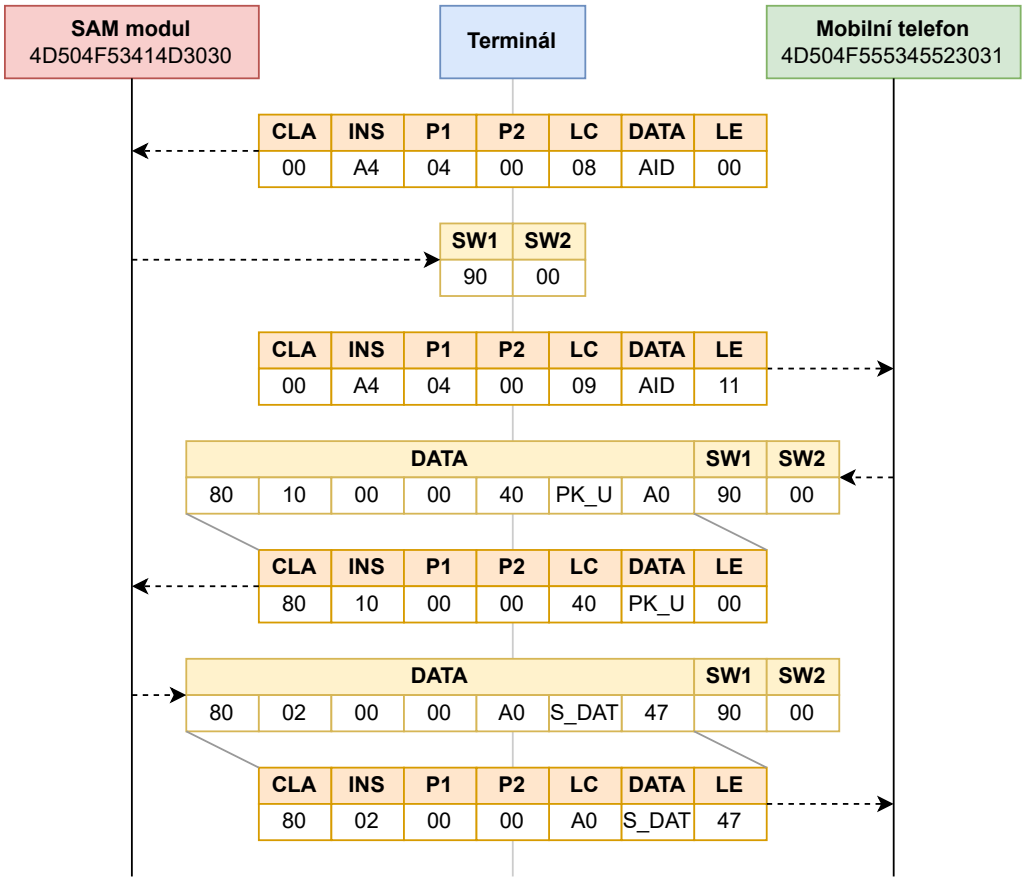
Obr. 3.2: Struktura autentizačního časového tokenu.

3.3 Zapouzdřovaní APDU zpráv

Komunikační model APDU sám neumožňuje komunikovat jiným způsobem než kombinací zpráv požadavek-odpověď. Přímou příčinou je, že k vytvoření komunikačního kanálu mezi mobilním telefonem a chytrou kartou tímto standardním způsobem je vyžadován mezilehlý uzel implementující logiku předávání dat jednotlivých APDU transakcí. Lze však implementovat protokol tak, aby mezilehlý uzel disponoval pouze omezenou logikou. Tu lze provádět například na mikrokontroléru, který by však stačila k předávání zpráv mezi zařízeními. K vytvoření spojení pak stačí uzlu vybrat aplikace APDU na koncových zařízeních a následně univerzálním způsobem předávat přijímané zprávy.

Zapouzdření APDU zpráv, jak je ukázáno na obrázku 3.3, staví na přípravě APDU požadavku již na koncovém zařízení. Takto připravenou zprávu lze uložit do datového pole APDU odpovědi, ze kterého je extrahována na mezilehlém uzlu. Ten zprávu může poslat na druhé koncové zařízení, jež může opět odpovědět připravenou APDU zprávou v datovém poli odpovědi.

Nejdříve se tedy připojí SAM modul s přesně předdefinovanou hodnotou AID aplikace. Vybere se příslušná aplikace, a to pomocí APDU SELECT příkazu. Předdefinovanou hodnotu AID má také druhé koncové zařízení, tj. například mobilní aplikace. Po vyslání zprávy APDU SELECT na toto zařízení jsou již v odpovědi na SELECT k dispozici data nové APDU zprávy, která je odeslána na SAM modul. Ten data zpracuje, vykoná implementované instrukce a do odpovědi připraví novou APDU zprávu. Tímto způsobem lze neustále předávat data, dokud například odpověď neobsahuje sekvence hodnot indikující konec komunikace. Dalšími varianty přerušující tok dat jsou chyby v přenosu nebo příjem stanoveného kódu ohlašujícího chybu na koncovém zařízení. Využití této metody vede k univerzálnosti celého řešení.



Obr. 3.3: Ukázka metody zapouzdřování APDU zpráv.

Díky tomu mohou být jednotlivé elementy systému opakovaně použity, upravovány a jednoduše rozšiřovány o další funkce.

3.4 Způsob sestavení a spuštění

Využití implementovaného protokolu vyžaduje v prvé řadě zjištění připojených periferií v podobě čteček čipových karet a NFC rozhraní. Proto slouží aplikace `pcsc-list`, kterou je potřeba nejdřív sestavit, a to například pomocí skriptu `debug.sh` (tento skript aplikaci automaticky také spustí). Ze seznamu připojených čteček, který může vypadat jako na výpisu 3.1, lze následně vybrat, které budou používané v protokolu (zkopírováním názvů - celý řádek).

Výpis 3.1: Ukázka výstupu z aplikace pcsc-list.

```
Listing all connected PC/SC readers:
HID Global OMNIKEY 3x21 Smart Card Reader [OMNIKEY 3x21
    ↪ Smart Card Reader] 00 00
ACS ACR1251 Dual Reader [ACR1251 1S CL Reader PICC] 01 00
Done!
```

Výpis 3.2: Ukázka konfigurace výběru PC/SC rozhraní pro aplikace.

```
#define VERIFIER_READER "HID Global OMNIKEY 3x21 Smart
    ↪ Card Reader [OMNIKEY 3x21 Smart Card Reader] 00 00"
#define USER_READER "ACS ACR1251 Dual Reader [ACR1251 1S
    ↪ CL Reader PICC] 01 00"
```

Názvy čteček lze následně vepsat ve složce aplikace `asaka-security-scheme` do souboru `config/config.h` k definovaným hodnotám `USER_READER` (např. NFC pro komunikaci s mobilním telefonem) a `VERIFIER_READER` (čtečka s připojenou MultOS kartou), jak je ukázáno na výpisu 3.2. Aplikaci lze sestavit a spustit pomocí skriptu `debug.sh` nebo `release.sh` (podle typu sestavení) ve složce `scripts`. Prostřednictvím nástroje CMAKE, jehož rozhraní je k dispozici přímo v rámci vývojového prostředí, je možné aplikaci sestavit pomocí konfigurace `security-scheme-android`.

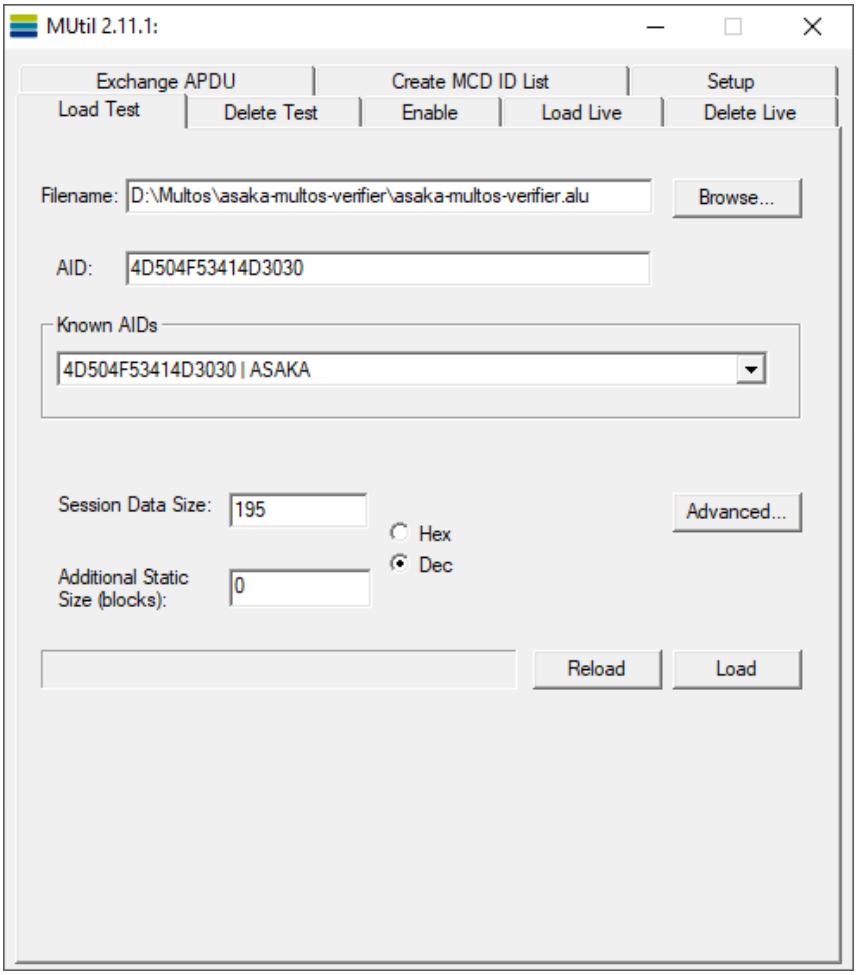
3.5 MultOS ověřovatel

Čipová karta MultOS může být použita jako SAM modul k provádění ověřovací části protokolu. Kód pro kartu (`asaka-MultOS-verifier`) musí být sestaven a nahrán na kartu pomocí nástrojů MultOS Smart Deck. Pro sestavení lze použít skript `build.bat`, který i vypíše paměťové parametry aplikace. Parametr `.DB` (tzv. `Session RAM`) je pak vstupním parametrem pro aplikaci `MUtil.exe`, která slouží pro nahrání sestavené aplikace na čipovou kartu. Aplikace se nahrává na čipovou kartu stejným způsobem, a to pomocí nástroje `MUtil.exe`. Hexadecimální hodnota identifikátoru aplikace čipové karty je nastavena na `4D 50 4F 53 41 4D 30 30`, a podle toho je potřeba nastavit parametry nahrávání aplikace do paměti karty. Struktura paměti sestavené aplikace je pak vidět v tabulce 3.1 a nastavení aplikace `MUtil.exe` na obrázku 3.4.

Data, která karta přijme APDU požadavkem, se ukládají do globální proměnné v hlavním souboru `main.c`. Po propsání těchto dat se vstupuje do funkce `main`, která slouží pro zpracování a přípravu odpovědi. V této funkci jsou definované i implementované všechny APDU instrukce, které má karta znát. Při vstupu do implementace konkrétní APDU instrukce, karta vykoná naprogramovaný kód a do stejného pole,

Tab. 3.1: Rozložení paměti aplikace protokolu ASAKA pro čipové karty MultOS.

start	stop	size	decimal	name
00000000	000000c2	c3	195	.DB
00000000	000000ff	100	256	.PB
00000000	00000474	45e	1118	.SB
00000000	00000464	4b8	1208	.text



Obr. 3.4: Nahrávání appletu ASAKA protokolu na čipovou kartu.

které sloužilo pro uložení příchozího požadavku, uloží odpověď. Ta je následně odeslána.

Implementace operací na eliptické křivce má své specifické vlastnosti. Na čipových kartách MultOS se bod na eliptické křivce ukládá jako posloupnost bajtů, a to ve struktuře, která je znázorněna na obrázku 3.5. Tato struktura odpovídá použité křivce `secp256k1`, což je 256 b křivka, jejíž bod je vyjádřen souřadnicemi o celkové délce 64 B. Kromě souřadnic potřebuje MultOS uvést tuto datovou strukturu for-

mátem bodu (v tomto případě formát zápisu odpovídá hodnotě `0x04`). Data bodu eliptické křivky se v této implementaci přenášejí bez úvodního bajtu informujícího o formátu a přenášejí se pouze souřadnice. Při využití například *micro-ecc* knihovny pro práci s body, které byly získány z MultOS karty prostřednictvím APDU odpovědi, je potřeba hodnoty souřadnic prohodit. Stejně tak v tomto případě musejí být souřadnice prohozeny i při přenosu jejich dat na čipovou kartu.

Bod eliptické křivky (256 b)		
Formát bodu	**Souřadnice X**	**Souřadnice Y**
1 B	32 B	32 B

Obr. 3.5: Struktura bodu eliptické křivky v paměti čipové karty MultOS.

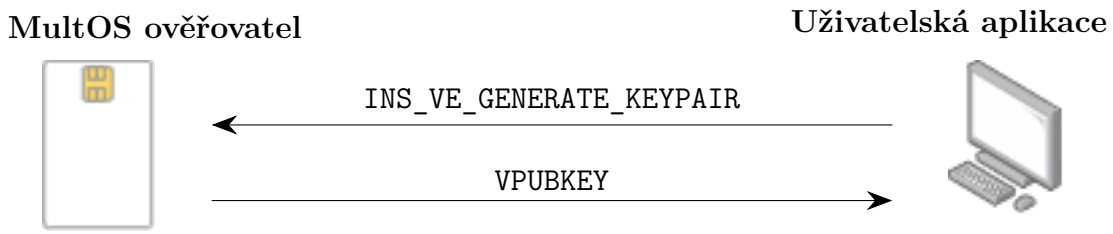
Reprezentace hodnot skaláru v paměti čipové karty MultOS je vyjádřeno polem bajtů o délce odpovídající vybrané eliptické křivce. Před samotnými daty skaláru je přidán jeden vodící bajt, který potřebuje čipová karta pro zpracování dat. Struktura je vidět na obrázku 3.6. Během přenosu přes APDU zprávy není vodící bajt přenášen.

Skalár (256 b)	
Vodící bajt	**Data skaláru**
1 B	32 B

Obr. 3.6: Struktura bodu eliptické křivky v paměti čipové karty MultOS.

3.5.1 Proces personalizace

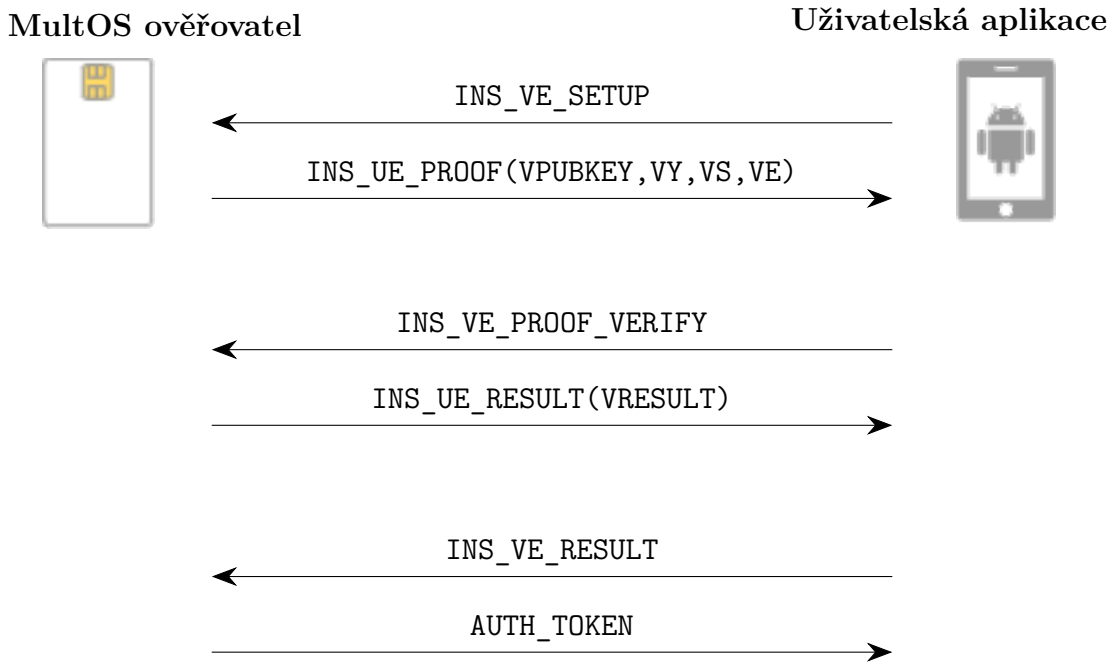
Během procesu personalizace entity MultOS ověřovatel dochází k vytvoření páru klíčů, který je uložen do statické paměti čipové karty. K tomu slouží implementovaná instrukce `INS_VE_GENERATE_KEYPAIR`. Během této fáze čipová karta vrací v odpovědi svůj veřejný klíč. Přepsání klíčů je možné pouze při opětovném zavolání této funkce. Komunikace během procesu personalizace je vykreslena na obrázku 3.7. Proces personalizace čipové karty je potřeba spustit pouze jednou.



Obr. 3.7: Sled zpráv personalizace protokolu ASAKA na čipové kartě.

3.5.2 Proces autentizace

Protokol je zahájen požadavkem `INS_VE_SETUP`, kterou je vygenerován závazek ověřovatele. V odpovědi na inicializační zprávu je zaslána připravená zpráva `INS_UE_PROOF`. Tu lze předat přímo uživateli. Samotné ověření uživatele je provedeno při zpracování zprávy `INS_VE_PROOF_VERIFY`. Poslední je zpráva `INS_VE_RESULT`, která vrátí autentizační časový token. Současně je také odvozen společný klíč. Celý sled zpráv je vykreslen na obrázku 3.8.



Obr. 3.8: Sled zpráv autentizace protokolu ASAKA na čipové kartě.

3.5.3 Dostupné APDU instrukce

Pomocí APDU zpráv lze volat tyto implementované funkce:

1. `INS_VE_GENERATE_KEYPAIR` (`0x00`): požadavek na vygenerování páru klíčů ověřovatele a předání veřejného klíče, na který čipová karta odpovídá daty veřejného klíče o délce 64 B v podobě bodu na eliptické křivce,

APDU požadavek				
CLA	**INS**	**P1**	**P2**	**LE**
0x80	0x00	0x00	0x00	0x40

APDU odpověď		
DATA	**SW1**	**SW2**
VPUBKEY	0x90	0x00

2. `INS_VE_SETUP` (`0x10`): požadavek na nastavení všech základních parametrů pro ověřovatele v dané relaci a výpočet závazku ověřovatele, který je posléze odeslán zapouzdřený v datech APDU odpovědi,

APDU požadavek						
CLA	**INS**	**P1**	**P2**	**LC**	**DATA**	**LE**
0x80	0x10	0x00	0x00	0x40	UPUBKEY	0xA0

APDU odpověď		
DATA	**SW1**	**SW2**
INS_UE_PROOF(VPUBKEY,VY,VS,VE)	0x90	0x00

3. `INS_VE_PROOF_VERIFY` (`0x20`): ověření uživatele (úspěšně ověření značí nastavení bajtu `VRESULT` na hodnotu `1`) a výpočet sdíleného klíče relace,

APDU požadavek						
CLA	**INS**	**P1**	**P2**	**LC**	**DATA**	**LE**
0x80	0x20	0x00	0x00	0x41	UE,US	65

APDU odp.		
DATA	**SW1**	**SW2**
INS_UE_RESULT(VRESULT)	0x90	0x00

4. `INS_VE_RESULT` (`0x30`): dešifruje získaný token od uživatele a ověří jeho správnost pomocí úvodní série sedmi bajtů, které mají nabývat hodnot `0` až `6`, a po nich jsou čtyři bajty kódující aktuální datum.

APDU požadavek						
CLA	**INS**	**P1**	**P2**	**LC**	**DATA**	**LE**
0x80	0x30	0x00	0x00	0x10	UTOKEN	0x0B

APDU odp.		
DATA	**SW1**	**SW2**
AUTH_TOKEN	0x90	0x01

3.6 Uživatelská aplikace

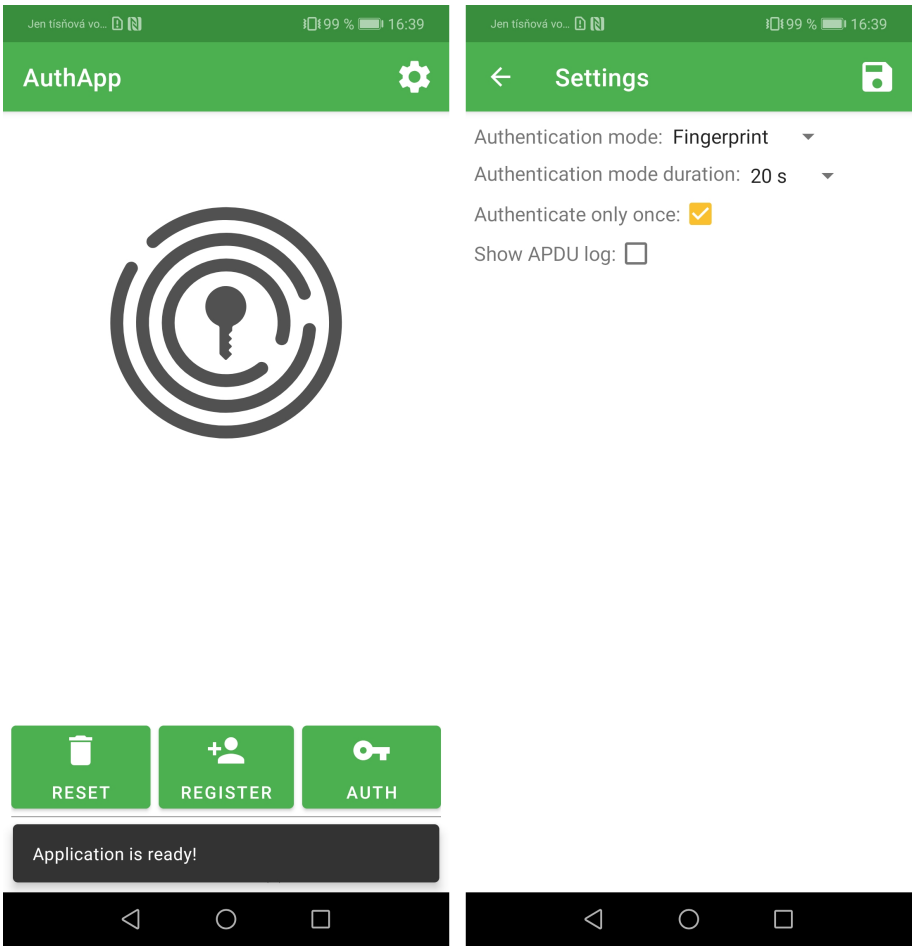
Kryptografické jádro uživatelské aplikace, které obsahuje implementaci funkcionalit protokolu, je napsané v jazyce C++. Tyto funkce jsou volány ze standardní aplikace naprogramované v jazyce Java a tato aplikace také spravuje ukládání hodnot do paměti. Více o využití Android NDK lze nalézt v části Vývoj aplikací pro operační systém Google Android. Pro správné fungování aplikace je potřeba mít nainstalovanou verzi operačního systému Google Android 5.0 (a novější) a zařízení musí podporovat rozhraní NFC i Bluetooth. Aplikace je rozdělena na dva logické balíčky, a to `app` pro mobilní telefony a `wear` pro chytré hodinky. Pro serializaci dat, tj. uchovávání stavu objektů, je využívána knihovna `Gson` a pro animaci uživatelského prostředí knihovna `Lottie` [42] [43]. Aplikaci lze nainstalovat dvěma způsoby:

- **pomocí souboru se sestavenou aplikací**: nainstalováním archivu `APK` se sestavenou aplikací na zařízení,
- **přes USB rozhraní**: nainstalováním aplikace přímo z vývojového prostředí Android Studio na zařízení.

3.6.1 Část aplikace pro mobilní telefon

Grafické rozhraní aplikace mobilního v aktuální podobě je vidět na obrázku 3.9. Aplikace je rozdělena do dvou obrazovek. Hlavní obrazovka zpřístupňuje hlavní prvky funkcionalit aplikace. Dále pak umožňuje přejít do nastavení aplikace. V dolní části nabízí informační oblast, pomocí které se může uživatel dozvědět o událostech v rámci fungování aplikace, jako může být třeba informace o absenci rozhraní NFC nebo neúspěšné autentizaci.

Proces registrace vytváří zcela nový objekt uživatele. Tento objekt je pak následně uložen do paměti mobilního telefonu. Při registraci je uživatel vyzván k zadání kódu PIN, který může být následně použit během samotného procesu autentizace.



Obr. 3.9: Hlavní obrazovka (vlevo) a možnosti nastavení (vpravo) uživatelské aplikace.

V případě, že si uživatel přeje vymazat svoje údaje, může tak učinit přes tlačítko pro reset dat aplikace. Reset je proveden úplným odstraněním uživatelských dat z paměti mobilního telefonu.

Aplikaci je možné přepnout do režimu autentizace, přičemž uživatel je ověřen pomocí kódu PIN, který reprezentuje faktor znalosti, nebo otisku prstu, který reprezentuje faktor vlastnosti entity. Jakým způsobem bude uživatel ověřen je určeno nastavením aplikace, jež je možné změnit na samostatné obrazovce. Při úspěšném ověření identity uživatele je aplikace přepnuta do autentizačního režimu. V tomto režimu je aplikace po omezenou dobu schopná spustit implementovaný protokol a komunikovat prostřednictvím rozhraní NFC i Bluetooth.

Obrazovka nastavení umožňuje uživateli nastavit tyto parametry:

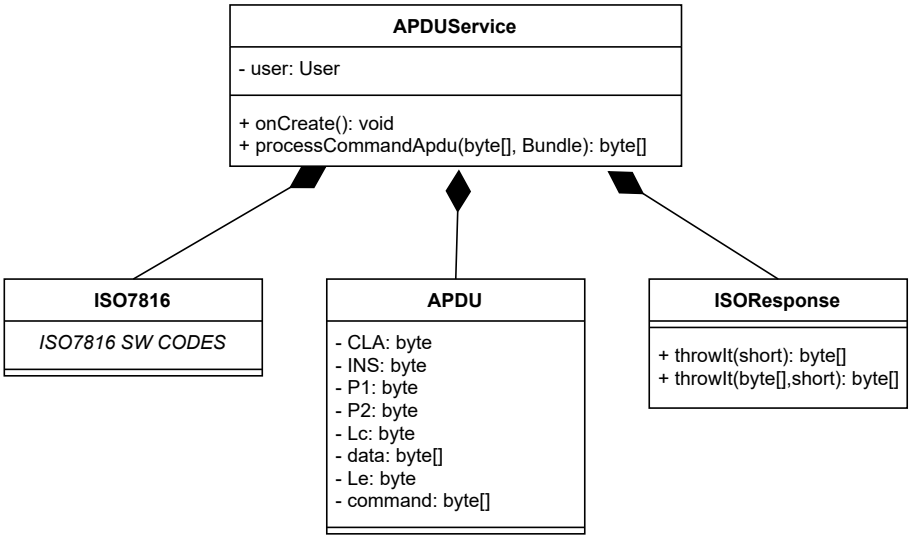
1. **způsob autentizace** (angl. *Authentication mode*): určuje, jakým faktorem bude uživatel ověřen, tj. otisk prstu nebo PIN kód,
2. **časové omezení režimu autentizace** (angl. *Authentication mode duration*):

nastavuje časové omezení autentizačního režimu aplikace, tj. časového okna, ve kterém může uživatelská aplikace komunikovat s ověřovatelem,

3. **možnost jednorázového autentizačního módu** (angl. *Authenticate only once*): v případě aktivovaného nastavení je autentizační mód deaktivován v momentě dokončení prvního průchodu protokolem, a to nehledě na nastavený časový limit,
4. **zobrazení APDU zpráv** (angl. *Show APDU log*): nastavuje, jestli mají být na hlavní obrazovce aplikace vypisovány přenášené APDU zprávy, což je vhodné například při ladění implementace protokolu.

Obsluha APDU zpráv

Hlavní třída, která se stará o obsluhu příchozích APDU zpráv je třída `APDUService`. Ta využívá `ISOResponse` pro obsluhu odpovědí a pomocnou třídu `APDU` pro jednodušší práci s obsahem APDU zpráv. Celá struktura je vidět na obrázku 3.10

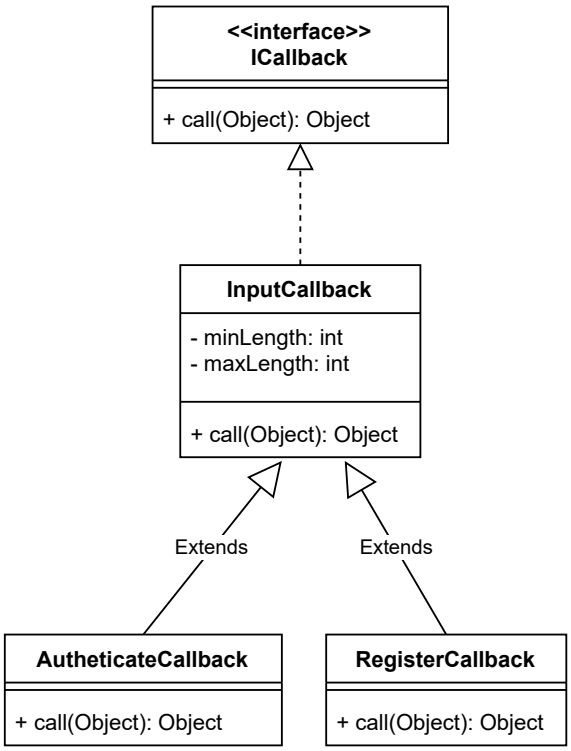


Obr. 3.10: Vrstva obsluhy APDU zpráv uživatelské aplikace.

Mechanizmus zpětného volání

Aplikace využívá mechanizmu zpětného volání (angl. *callback*) po vykonání určité činnosti. Rozhraní `ICallback` definuje jednu veřejnou metodu, kterou lze implementovat dle požadavků konkrétní činnosti. Toto je využíváno například pro práci s daty po úspěšné autentizaci nebo v případě registrace uživatele do aplikace. Struktura implementace je znázorněna na obrázku 3.11. Navržená struktura umožňuje využít dědičnosti a implementaci nových tříd zpětného volání, které rozvíjí již existující.

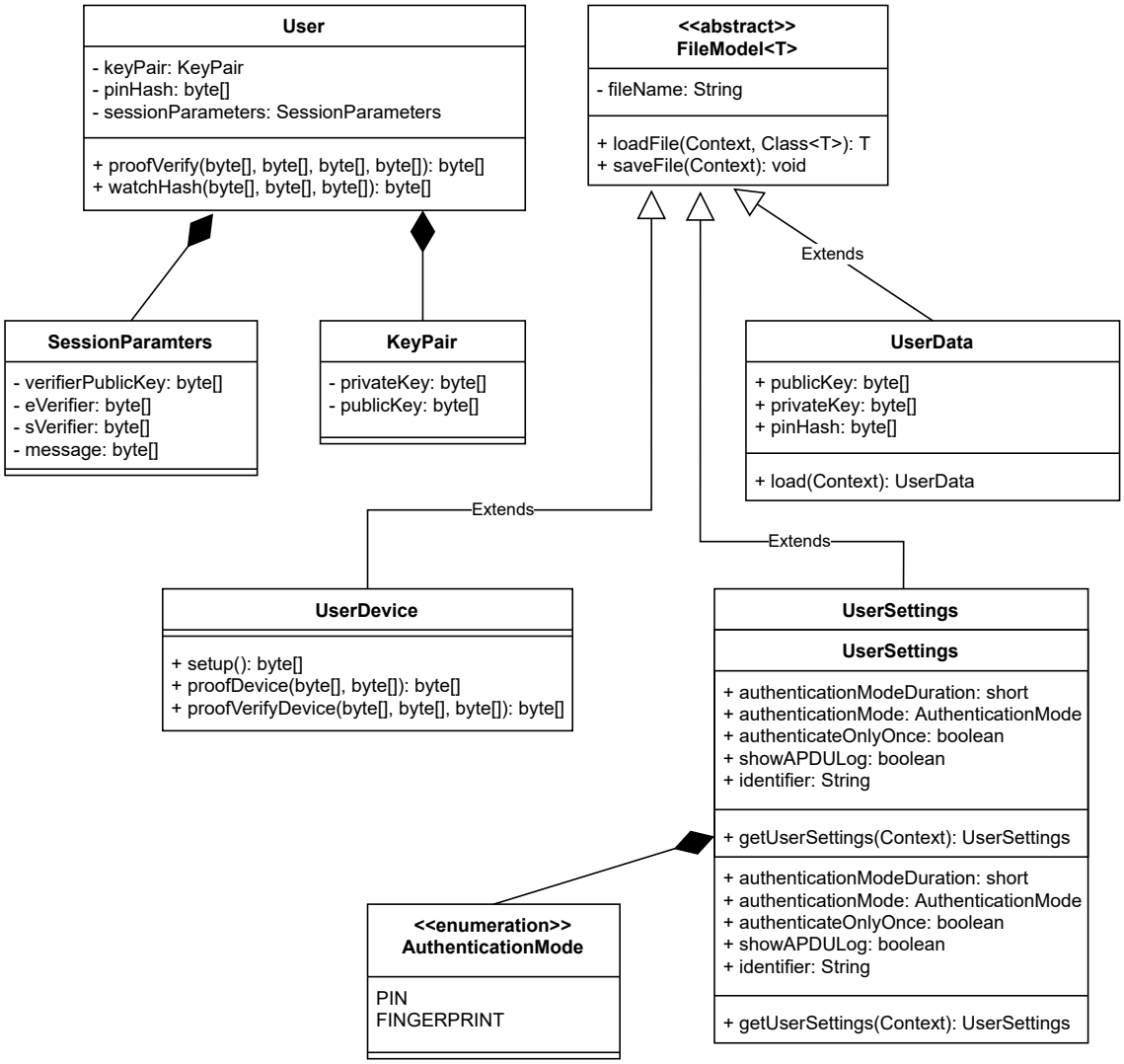
Příkladem mohou být třídy `AuthenticateCallback` a `RegisterCallback` se společným předkem `InputCallback`. Společné chování je v obou případech vstup dat z grafického rozhraní a ten řeší právě `InputCallback`.



Obr. 3.11: Implementace zpětného volání v rámci uživatelské aplikace.

Datová vrstva

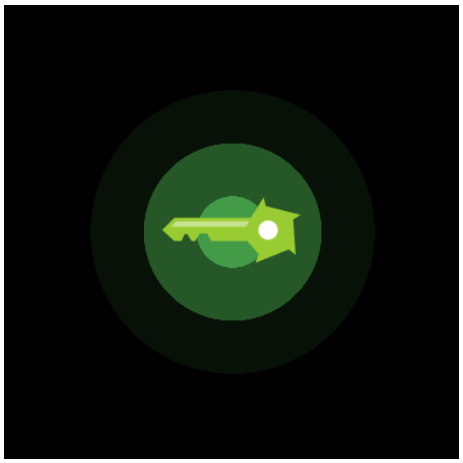
Datová vrstva, která je znázorněná na obrázku 3.12, má připravené mechanismy po práci se soubory a serializaci objektů. Abstraktní generická třída `FileModel<T>` umožňuje toto implementovat jakékoliv jiné třídě. Jako generický parametr je předávána právě implementující třída. Ve zdrojových kódech tuto třídu využívají `UserData`, pro ukládání registračních dat uživatele, `UserSettings` pro ukládání nastavení uživatele a `UserDevice`, pro data externího zařízení uživatele. Dále v rámci datové vrstvy je třída User, která reprezentuje načtená data uživatele po úspěšné autentizaci. Tato třída je používána například ve službě obsluhující APDU.



Obr. 3.12: Datová vrstva uživatelské aplikace.

3.6.2 Část aplikace pro chytré hodinky

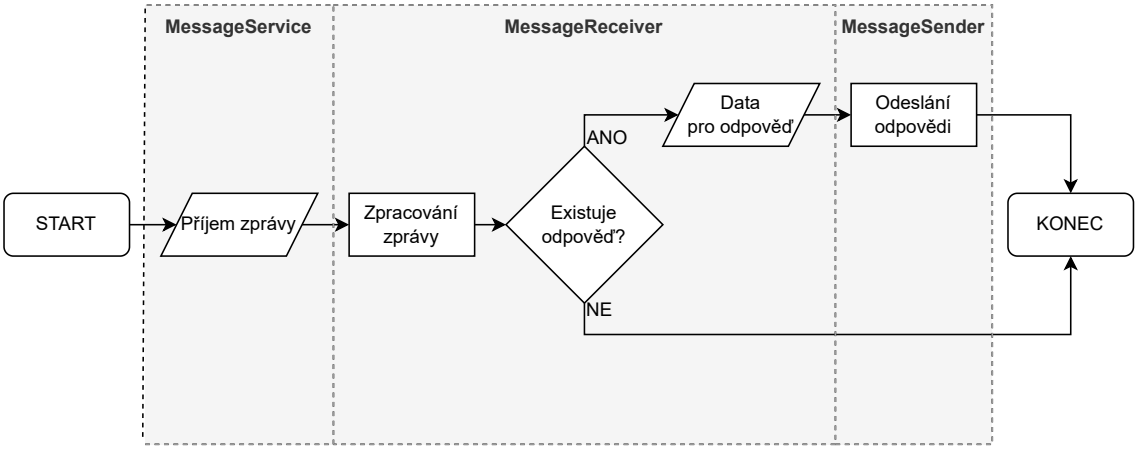
Část aplikace pro chytré hodinky je jednodušší, než část pro mobilní telefon. Je nutné, aby chytré hodinky a mobilní telefon byly spárovány a propojeny technologií Bluetooth, což umožňuje vytvoření komunikačního kanálu a předávání zpráv s instrukcemi a aplikačními daty. Ihned po spuštění aplikace na hodinkách se spustí komunikační rozhraní aplikace i funkce potřebné pro implementovaný protokol. Grafické rozhraní bylo v tomto případě vytvořeno jednoduše, aby uživatel viděl pomocí animovaného indikátoru, že aplikace je připravená a běží, jak je vidět na obrázku 3.13. Implementace této části je vytvořena jako balíček vedle aplikace pro mobilní telefon a implementuje pouze nejnutnější prvky, aby bylo šetřeno prostředky.



Obr. 3.13: Obrazovka uživatelské aplikace na chytrých hodinkách.

3.6.3 Komunikace mezi mobilním telefonem a chytrými hodinkami

Datové spojení mezi mobilním telefonem a chytrými hodinkami funguje na technologii Bluetooth. Zařízení musejí být spárovaná, aby spolu mohla komunikovat. Rozhraní k přenosu dat jsou odvozena od existující implementace pana Klasovitého [44]. Implementace byla ale upravena pro potřeby této práce. Komunikační komponenty byly vyčleněny do jednotlivých tříd a souborů, které lze následně používat univerzálně napříč zdrojovými kódy.



Obr. 3.14: Vývojový diagram implementace zpracování zpráv na zařízení Google Android.

Komunikace funguje na principu předávání zpráv prostřednictvím sdíleného API operačního systému Google Android mezi chytrými zařízeními. Na obou zařízeních

běží části pro čtení a odesílání zpráv. Konkrétní chování při přijetí zpráv je pak definováno přímo v kódu. Jaké chování má v aplikaci nastat indikuje konkrétní parametr zprávy, podobně jako parametr pro instrukce na čipových kartách. Komunikace je řešena asynchronně, jednotlivé transakce zpráv proto neblokují chod aplikace. Přenášená data jsou zpracovávána jako pole bajtů.

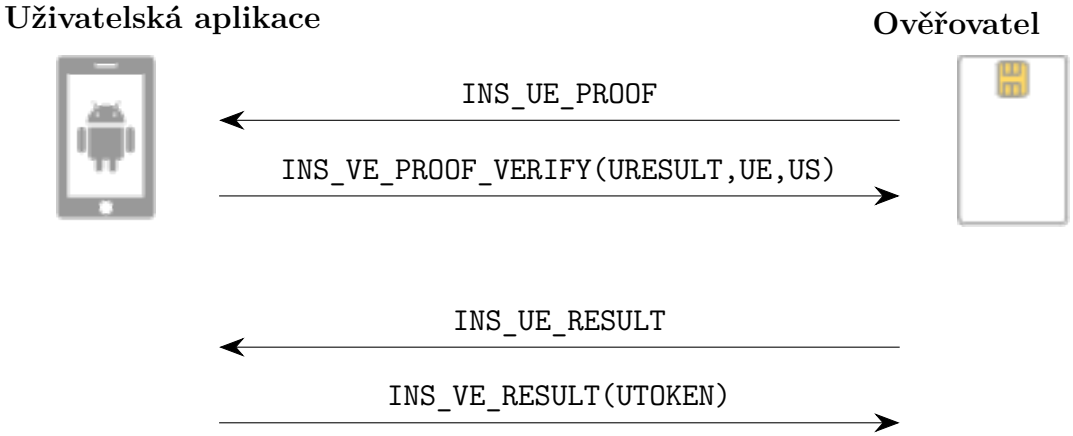
Kvůli asynchronnímu volání je nutné informovat terminál (tj. uzel spojující ověřovatele a uživatele) o stavu zpracování. Proto byl využit vlastní stavový kód s hodnotami `0x89 0x00`. Z případě, že je tento kód přijat na mezilehlém uzlu, přenos APDU požadavku na mobilní telefon se opakuje. To se děje do momentu, kdy je přijata zpráva s kódem `0x90 0x00`, která indikuje úspěšné zpracování.

Řetězec zpracování zpráv je vidět na obrázku 3.14 a je složen ze tří tříd obstarávajících komunikaci:

1. `MessageService.java`: třída služby pro registraci příchozích zpráv v systému,
2. `MessageReceiver.java`: třída zpracovávající příchozí zprávy a volající implementované chování,
3. `MessageSender.java`: třída pro odesílání zpráv s možností přiložit data.

3.6.4 Komunikace uživatelské aplikace s ověřovatelem

Pakliže uživatel úspěšně ověřil protistranu, pak o této skutečnosti předává informaci prostřednictvím zprávy `INS_VE_PROOF_VERIFY`, a to v prvním bajtu datové části (bajt s označením `URESULT`). Pokud je hodnota tohoto bajtu rovna `1`, ověřil uživatel protistranu. V té stejné odpovědi předává také vlastní závazek, pomocí kterého pak dojde k ověření na straně ověřovatele a výpočtu společného klíče. Pomocí zjištěného klíče lze přenést zašifrovaný autentizační token v rámci transakce `INS_UE_RESULT`. V jakém pořadí jsou zprávy posílány uživateli je vidět na obrázku 3.15.



Obr. 3.15: Sled zpráv protokolu ASAKA na mobilním telefonu.

Dostupné APDU instrukce

Uživatelská aplikace má podporu APDU zpráv:

1. `INS_UE_PROOF` (`0x02`): ověření závazku ověřovatele uživatelem (jestli byl ověřovatel úspěšně ověřen na straně uživatele, pak je bajt `URESULT` v odpovědi nastaven na hodnotu `1`) a získání závazku uživatele,

APDU požadavek						
CLA	**INS**	**P1**	**P2**	**LC**	**DATA**	**LE**
0x80	0x02	0x00	0x00	0xA0	VPUBKEY,VY,VS,VE	0x41

APDU odpověď		
DATA	**SW1**	**SW2**
INS_VE_PROOF_VERIFY(URESULT,UE,US)	0x90	0x00

2. `INS_UE_RESULT` (`0x03`): v případě úspěšné autentizace indikované hodnotou `1` v datech `VRESULT` odpoví APDU zprávou s časovým tokenem zašifrovaným smluveným klíčem šifrou 3DES.

APDU požadavek						
CLA	**INS**	**P1**	**P2**	**LC**	**DATA**	**LE**
0x80	0x03	0x00	0x00	0x01	VRESULT	0x10

APDU odpověď		
DATA	**SW1**	**SW2**
INS_VE_RESULT(UTOKEN)	0x90	0x00

3.7 Výkonové testy a funkční testování

Po implementaci jednotlivých částí a jejich otestování, byla provedena série deseti měřených průchodů, jejichž výsledky jsou zaznamenány v tabulce 3.2. Ke všem průchodům byla použita stejná zařízení, a to mobilní telefon HUAWEI P20 Lite, chytré hodinky HUAWEI Watch 2 a čipová karta MultOS s označením `ML4-G21`. Snímek propojených zařízení se spuštěnou aplikací v autentizačním režimu lze vidět na obrázku 3.16. Jako prvek pro zprostředkování spojení mobilního telefon a čipové karty bylo využito počítače Raspberry Pi Model 3 B+. Během měření byly mobilní telefon i chytré hodinky odpojeny od sítě Wi-Fi, v popředí nebyly spuštěny žádné další aplikace a jejich výkon nebyl nijak omezen (například režimem úspory baterie na úrovni operačního systému). Mobilní telefon používá osmijádrový procesor Kirin 695 taktovaný na 2,36 GHz, zatímco hodinky využívají čipovou sadu Qualcomm MSM8909W Snapdragon Wear 2100, která integruje čtyři jádra ARM Cortex-A7

s taktem 1,1 GHz. Naměřené hodnoty zahrnují také čas pro komunikaci a režii, protože u čipových karet není možné změřit čas na procesorovém jádře.

Tab. 3.2: Výsledky měření výkonu implementovaného protokolu.

Průchod	1.	2.	3.	4.	5.	6.	7.	8.	9.	10.
MultOS (s)	2,90	2,91	2,90	2,91	2,90	2,89	2,90	2,90	2,89	2,90
Android (s)	0,34	0,36	0,38	0,38	0,33	0,36	0,36	0,38	0,38	0,37
Celkem (s)	3,25	3,27	3,29	3,30	3,24	3,26	3,27	3,29	3,28	3,28
Odchyl. (s)	0,02	0,00	-0,02	-0,03	0,03	0,02	0,01	-0,02	-0,01	-0,01

Zpracování všech naměřených hodnot ukazuje, že průměrný čas potřebný pro entitu ověřovatele na platformě MultOS je 2,9 s, pro uživatele je to 0,37 s a tedy pro celkový běh protokolu 3,27 s včetně spuštěné části na chytrých hodinkách. Mezi jednotlivými průchody měření jsou drobné odchylky, jak je vidět na posledním řádku tabulky. Příčinnou odchylek může být například použité NFC rozhraní na straně mobilního telefonu nebo přenos zpráv bezdrátovou technologií Bluetooth. Časově náročná je také režie APDU zpráv a jejich přenos. Záznam komunikace mezi ověřovatelem a uživatelskou aplikací je vložen do přílohy C.

Obr. 3.16: Mobilní telefon a chytré hodinky se spuštěnou uživatelskou aplikací.

Závěr

V rámci závěrečné práce byl implementován AKA protokol s použitím čipové karty platformy MultOS. Čipová karta jako SAM modul zvyšuje úroveň zabezpečení systému. Na MultOS čipových kartách nelze používat žádné externí knihovny, proto bylo využito integrovaných funkcí. Kód pro karty je napsaný především v jazyce C, ale konkrétní kryptografické funkce jsou napsány v jazyce symbolických adres. Jako mezivrstva zprostředkující komunikaci čipové karty s mobilním telefonem, bylo využito zařízení Raspberry Pi. Software i v tomto případě byl napsán v jazyce C, je ale konfigurovatelný tak, že sám umí emulovat roli uživatele. Toho lze využít při vývoji appletu pro kartu. Do systému vstupuje uživatel pomocí aplikace v mobilním telefonu s operačním systémem Google Android pomocí NFC a chytrými hodinkami pomocí rozhraní Bluetooth. Aplikace mobilního telefonu umožňuje autentizovat uživatele faktory znalosti (tj. PIN kód) nebo vlastnosti (tj. otisku prstu), vlastnictví (tj. mobilní telefon). Dalším faktorem vlastnictví jsou pak spárované chytré hodinky, na nichž běží další část aplikace. Pro platformu Google Android bylo naprogramováno kryptografické jádro v jazyce C++ prostřednictvím nástrojové sady Android NDK. Jako knihovna pro kryptografii na eliptických křivkách byla na vybrána volně dostupná knihovna `micro-ecc`. Komunikační model mezi telefonem a čipovou kartou využívá metody zapouzdřování APDU zpráv, což je způsob, jakým lze přemostit spojení dvou zařízení podporujících tento protokol bez dodatečné logiky mezilehlého bodu. Toho lze například využít při implementaci systému prostřednictvím jednoduchého přístupového terminálu.

Kód pro čipové karty je zdokumentovaný a není vázaný na tuto implementaci jako celek, proto je možné jeho využití i v dalších projektech. Kryptografické jádro aplikace pro Google Android má připravené univerzální funkce, které lze využít obdobně v jiných aplikacích. Aplikace v tuto chvíli podporuje využití pouze jednoho zařízení (chytrých hodinek) jako dalšího faktoru vlastnictví, nicméně by bylo možné systém rozšířit o další chytrá zařízení, která by se na autentizaci uživatele mohla podílet.

Výsledky diplomové práce byly publikovány na studentské konferenci EEICT 2022 v článku *Access control system using multi-factor authentication*, který se umístil na druhém místě v kategorii *Network Security and Cryptography* [45].

Literatura

[1] Ministerstvo průmyslu a obchodu České republiky. [online]. [cit. 1. 12. 2021]. URL: https://www.mpo.cz/assets/dokumenty/53723/64358/658713/priloha001.pdf.

[2] Secure access module. [online]. [cit. 11. 11. 2021]. URL: https://www.cardlogix.com/glossary/sam-card-secure-access-module-secure-application-module/.

[3] Access Control Models. [online]. [cit. 1. 12. 2021]. URL: https://westoahu.hawaii.edu/cyber/best-practices/best-practices-weekly-summaries/access-control/.

[4] Autentizace, 2020. [online]. [cit. 1. 12. 2021]. URL: https://cs.wikipedia.org/wiki/Autentizace.

[5] Authorization. [online]. [cit. 19. 5. 2022]. URL: https://www.techopedia.com/definition/10237/authorization.

[6] Petr Gašparík. Vícefaktorová autentizace. [online]. [cit. 1. 12. 2021]. URL: https://www.ami.cz/publikujeme/blog/vicefaktorova-autentizace.

[7] Commission Delegated Regulation (EU) 2018/389 of 27 November 2017 supplementing Directive (EU) 2015/2366 of the European Parliament and of the Council with regard to regulatory technical standards for strong customer authentication and common and secure open standards of communication (Text with EEA relevance.), Mar 2018. [online]. [cit. 12. 12. 2021]. URL: https://eur-lex.europa.eu/eli/reg_del/2018/389/oj.

[8] 370/2017 Sb. Zákon O Platebním Styku. [online]. [cit. 12. 12. 2021]. URL: https://www.zakonyprolidi.cz/cs/2017-370.

[9] Adrien Koutsos, 2019. [online]. [cit. 1. 12. 2021]. URL: https://ieeexplore.ieee.org/document/8806761.

[10] Jari Arkko and Henry Haverinen. Extensible Authentication Protocol Method for 3rd Generation Authentication and Key Agreement (EAP-AKA), 2006. [online]. [cit. 1. 12. 2021]. URL: https://datatracker.ietf.org/doc/html/rfc4187.

[11] Petr Dzurenda, Sara Ricci, Raúl Casanova Marqués, Jan Hajny, and Petr Cika. Secret sharing-based authenticated key agreement protocol. In *The 16th International Conference on Availability, Reliability and Security*, ARES 2021, New

York, NY, USA, 2021. Association for Computing Machinery. [online]. [cit. 4. 12. 2021]. doi:10.1145/3465481.3470057.

[12] Ben Lutkevich. What is a Microcontroller and How Does it Work?, 2019. [online]. [cit. 1. 12. 2021]. URL: https://internetofthingsagenda.techtarget.com/definition/microcontroller.

[13] Arduino. [online]. [cit. 1. 12. 2021]. URL: https://www.arduino.cc/.

[14] Tadeáš Cvrček. Kryptografie na platformě Arduino, 2020. [online]. [cit. 1. 12. 2021]. URL: http://hdl.handle.net/11012/190258.

[15] Raspberry Pi. Teach, Learn, and Make with Raspberry Pi. [online]. [cit. 7. 11. 2021]. URL: https://www.raspberrypi.org/.

[16] Raspberry pi documentation - raspberry pi pico. [online]. [cit. 19. 5. 2022]. URL: https://www.raspberrypi.com/documentation/microcontrollers/raspberry-pi-pico.html.

[17] Raspberry Pi 4 Model B specifications. [online]. [cit. 28. 11. 2021]. URL: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/.

[18] Smart card, 2021. [online]. [cit. 1. 12. 2021]. URL: https://en.wikipedia.org/wiki/Smart_card.

[19] Pc/sc workgroup. [online]. [cit. 1. 12. 2021]. URL: https://pcscworkgroup.com/.

[20] BasicCard. [online]. [cit. 25. 11. 2021]. URL: http://basiccard.com/.

[21] DotNet SmartCard. [online]. [cit. 25. 11. 2021]. URL: http://www.dotnetcard.net/.

[22] Petr Dzurenda. Kryptografická ochrana digitální identity, 2019. [online]. [cit. 1. 12. 2021]. URL: http://hdl.handle.net/11012/180699.

[23] APDU (Java Card API, Classic Edition), 2021. [online]. [cit. 25. 11. 2021]. URL: https://docs.oracle.com/en/java/javacard/3.1/jc_api_srvc/api_classic/javacard/framework/APDU.html.

[24] ISO7816 part 4 section 5 APDU level data structures. [online]. [cit. 26. 11. 2021]. URL: http://cardwerk.com/smart-card-standard-iso7816-4-section-5-basic-organizations.

[25] ISO/IEC 7816, 2021. [online]. [cit. 24. 11. 2021]. URL: https://en.wikipedia.org/wiki/ISO/IEC_7816.

[26] Multos Smartcard Technology, 2021. [online]. [cit. 21. 11. 2021]. URL: https://multos.com/technology/multos-smartcard-technology/.

[27] MULTOS smart card operating system chip card application development. [online]. [cit. 26. 11. 2021]. URL: https://cardwerk.com/multos-multi-application-smart-chip-card-operating-system/.

[28] Android Studio, 2021. [online]. [cit. 14. 11. 2021]. URL: https://en.wikipedia.org/wiki/Android_Studio.

[29] Android Studio. [online]. [cit. 14. 11. 2021]. URL: https://developer.android.com/studio/intro.

[30] The JIT compiler. [online]. [cit. 14. 11. 2021]. URL: https://www.ibm.com/docs/en/ztpf/1.1.0.15?topic=reference-jit-compiler.

[31] Android NDK: Get started with the NDK . [online]. [cit. 14. 11. 2021]. URL: https://developer.android.com/ndk/guides.

[32] Android NDK: JNI tips. [online]. [cit. 14. 11. 2021]. URL: https://developer.android.com/training/articles/perf-jni.

[33] Android NDK vs. Android SDK, 2017. [online]. [cit. 14. 11. 2021]. URL: http://androiddeveloper.galileo.edu/2017/03/08/android-ndk-vs-android-sdk/.

[34] Ieee 802.15 wpan task group 1 (tg1). [online]. [cit. 13. 5. 2022]. URL: https://www.ieee802.org/15/pub/TG1.html.

[35] Definition of bluetooth versions. [online]. [cit. 13. 5. 2022]. URL: https://www.pcmag.com/encyclopedia/term/bluetooth-versions.

[36] Bluetooth. [online]. [cit. 13. 5. 2022]. URL: https://en.wikipedia.org/wiki/Bluetooth.

[37] Definition of bluetooth le. [online]. [cit. 13. 5. 2022]. URL: https://www.pcmag.com/encyclopedia/term/bluetooth-le.

[38] Near-field communication. [online]. [cit. 19. 5. 2022]. URL: https://en.wikipedia.org/wiki/Near-field_communication.

[39] ISO/IEC 18092:2013. [online]. [cit. 19. 5. 2022]. URL: https://www.iso.org/standard/56692.html.

[40] Technical specifications - nfc forum. [online]. [cit. 19. 5. 2022]. URL: https://nfc-forum.org/our-work/specification-releases/specifications/nfc-forum-technical-specifications/.

[41] SHA1. [online]. [cit. 7. 11. 2021]. URL: https://github.com/clibs/sha1.

[42] Gson. [online]. [cit. 15. 5. 2022]. URL: https://github.com/google/gson.

[43] Lottie for android, ios, react native, web, and windows. [online]. [cit. 15. 5. 2022]. URL: https://github.com/airbnb/lottie-android.

[44] Kristián Klasovitý. Bezpečnostní systém využívající chytrá zařízení, 2021. [online]. [cit. 13. 5. 2022]. URL: https://www.vut.cz/studenti/zav-prace/detail/133525.

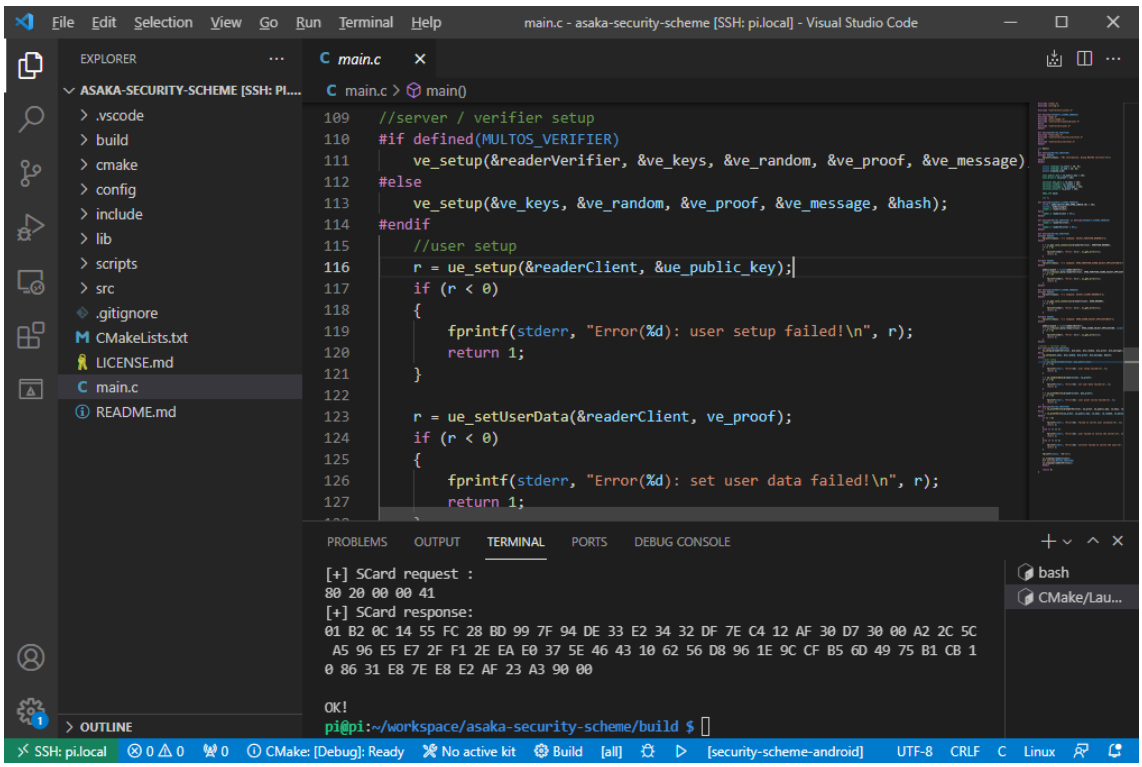
[45] Tadeáš Cvrček and Petr Dzurenda. Access control system using multi-factor authentication, 2022. [online]. [cit. 19. 5. 2022].

Seznam symbolů a zkratek

PSD2 SCA	Payment Services Directive 2 Strong Customer Authentication
ARM	Advanced RISC Machines
SAM	Secure Access Module
IoT	Internet of Things
AKA	Authentication and Key Agreement
DH	Diffie-Hellman
DSA	Digital Signature Algorithm
EAP	Extensible Authentication Protocol - Authentication and Key Agreement
AES	Advanced Encryption Standard
DES	Triple Data Encryption Algorithm
3DES	Triple Data Encryption Algorithm
ECB	Electronic Codebook
ASAKA	asymetrický AKA protokol
GPIO	General-Purpose Input/Output
HDMI	High-Definition Multimedia Interface
USB	Universal Serial Bus
PC/SC	Personal Computer/Smart Card
APDU	Application Protocol Data Unit
NFC	Near Field Communication
RTOS	Real-Time Operating System
SDK	Software Development Kit
JIT	Just-In-Time
NDK	Native Development Kit
JIN	Java Native Interface
IDE	Integrated Development Environment
BLE	Bluetooth Low Energy
SSH	Secure Shell

Seznam příloh

A Vývojové prostředí



B Struktura souborů implementace protokolu ASAKA

```
/ ..... kořenový adresář
├── build ..... adresář sestavených binárních souborů
├── source ..... adresář zdrojových souborů
│   ├── AuthApp ..... adresář aplikace pro Google Android
│   ├── asaka-multos-verififer ..... adresář kódu pro čipové karty MultOS
│   │   ├── include ..... adresář specifických hlavičkových souborů
│   │   ├── lib ..... adresář knihoven
│   │   └── src ..... zdrojové kódy
│   │       └── main.c ..... hlavní soubor
│   ├── pcsc-list ..... adresář nástroje pro výpis připojených PC/SC zařízení
│   │   ├── include
│   │   ├── lib
│   │   └── src
│   │       └── main.c
│   └── asaka-security-scheme ..... adresář aplikace pro Raspberry Pi
│       ├── include
│       ├── lib
│       └── src
│           └── main.c
└── README.pdf ..... popis úkonů potřebných ke spuštění
```

C Výpis komunikace protokolu ASAKA

```
[M] Information: Using MULTOS verifier!
[-] Command: SELECT_VERIFIER_READER
[!] Please insert a working reader...
[+] OK, using reader HID Global OMNIKEY 3x21 Smart Card Reader [OMNIKEY 3x21 Smart Card Reader] 00 00
[!] Waiting for card insertion...
[+] OK, connected!

[-] Command: APDU_VERIFIER_SCARD_SELECT_APPLICATION
[+] SCard request :
00 A4 04 00 08 4D 50 4F 53 41 4D 30 30 00
[+] SCard response:
90 00

V: 0.058286
[+] SCard request :
80 00 00 00 40
[+] SCard response:
4E 35 AD DE B0 EC 0A 3A 7F CE F4 41 E2 29 0A 4A 5E BA D4 AD 5B 61 9D FD 26 FD 23 C8 8F F7 EE 71 A3 65 65 29 18 94 89 5F B8 19 D2 7C A7 E4 D1 F7 AD 52 DF 18 89 21 09 E9 F8 46 13 B6 32 8B CC 18 90 00

[-] Command: SELECT_CLIENT_READER
[!] Please insert a working reader...
[+] OK, using reader ACS ACR1251 Dual Reader [ACR1251 1S CL Reader PICC] 01 00
[!] Waiting for card insertion...
[+] OK, connected!

[-] Command: APDU_SCARD_SELECT_APPLICATION
[+] SCard request :
00 A4 04 00 09 4D 50 4F 55 53 45 52 30 31 0B
[+] SCard response:
80 10 00 00 40 CA FF 1A D2 5B FD 32 1E 7F 52 30 2F 71 05 D9 1D 90 45 F8 56 F4 E7 58 40 AA 03 95 D8 27 05 49 F2 14 1C 9D 81 A7 D1 65 37 98 C3 90 93 00 54 9B 2A 49 6E B3 C1 9A 1C BC C0 25 CD C6 38 C8 71 54 E9 A6 90 00

C: 0.054104
[+] SCard request :
80 10 00 00 40 CA FF 1A D2 5B FD 32 1E 7F 52 30 2F 71 05 D9 1D 90 45 F8 56 F4 E7 58 40 AA 03 95 D8 27 05 49 F2 14 1C 9D 81 A7 D1 65 37 98 C3 90 93 00 54 9B 2A 49 6E B3 C1 9A 1C BC C0 25 CD C6 38 C8 71 54 E9 A6
[+] SCard response:
61 A6

[+] SCard request :
00 C0 00 00 A6
[+] SCard response:
80 02 00 00 A0 4E 35 AD DE B0 EC 0A 3A 7F CE F4 41 E2 29 0A 4A 5E BA D4 AD 5B 61 9D FD 26 FD 23 C8 8F F7 EE 71 A3 65 65 29 18 94 89 5F B8 19 D2 7C A7 E4 D1 F7 AD 52 DF 18 89 21 09 E9 F8 46 13 B6 32 8B CC 18 00 00 00 00 00 00 00 00 00 00 00 00 7F D5 E1 49 A2 32 29 59 BE A5 B9 04 23 AF 59 F4 13 87 FA 35 FD 68 9E 74 6A FA 6C 7B 17 09 88 26 4E FA F9 92 A1 AE AE 0A F9 BB 21 86 13 20 8B E5 E8 41 BF 53 00 CE F9 99 5A FA FF DC 62 82 0C AB 13 E4 56 C0 70 AD 84 9B BE 37 F7 D0 79 B8 2B 53 41 67 0E FE 47 90 00

V: 1.457762
[+] SCard request :
80 02 00 00 A0 4E 35 AD DE B0 EC 0A 3A 7F CE F4 41 E2 29 0A 4A 5E BA D4 AD 5B 61 9D FD 26 FD 23 C8 8F F7 EE 71 A3 65 65 29 18 94 89 5F B8 19 D2 7C A7 E4 D1 F7 AD 52 DF 18 89 21 09 E9 F8 46 13 B6 32 8B CC 18 00 00 00 00 00 00 00 00 00 00 00 00 7F D5 E1 49 A2 32 29 59 BE A5 B9 04 23 AF 59 F4 13 87 FA 35 FD 68 9E 74 6A FA 6C 7B 17 09 88 26 4E FA F9 92 A1 AE AE 0A F9 BB 21 86 13 20 8B E5 E8 41 BF 53 00 CE F9 99 5A FA FF DC 62 82 0C AB 13 E4 56 C0 70 AD 84 9B BE 37 F7 D0 79 B8 2B 53 41 67 0E FE
[+] SCard response:
89 00

C: 0.070629
[+] SCard request :
80 02 00 00 A0 4E 35 AD DE B0 EC 0A 3A 7F CE F4 41 E2 29 0A 4A 5E BA D4 AD 5B 61 9D FD 26 FD 23 C8 8F F7 EE 71 A3 65 65 29 18 94 89 5F B8 19 D2 7C A7 E4 D1 F7 AD 52 DF 18 89 21 09 E9 F8 46 13 B6 32 8B CC 18 00 00 00 00 00 00 00 00 00 00 00 00 7F D5 E1 49 A2 32 29 59 BE A5 B9 04 23 AF 59 F4 13 87 FA 35 FD 68 9E 74 6A FA 6C 7B 17 09 88 26 4E FA F9 92 A1 AE AE 0A F9 BB 21 86 13 20 8B E5 E8 41 BF 53 00 CE F9 99 5A FA FF DC 62 82 0C AB 13 E4 56 C0 70 AD 84 9B BE 37 F7 D0 79 B8 2B 53 41 67 0E FE
[+] SCard response:
89 00

```

```
C: 0.041907
[+] SCard request :
80 02 00 00 A0 4E 35 AD DE B0 EC 0A 3A 7F CE F4 41 E2 29 0A 4A 5E BA D4 AD 5B 61 9D FD 26 FD 23
    ↪ C8 8F F7 EE 71 A3 65 65 29 18 94 89 5F B8 19 D2 7C A7 E4 D1 F7 AD 52 DF 18 89 21 09 E9 F8
    ↪  46 13 B6 32 8B CC 18 00 00 00 00 00 00 00 00 00 00 00 00 7F D5 E1 49 A2 32 29 59 BE A5
    ↪ B9 04 23 AF 59 F4 13 87 FA 35 FD 68 9E 74 6A FA 6C 7B 17 09 88 26 4E FA F9 92 A1 AE AE 0A
    ↪  F9 BB 21 86 13 20 8B E5 E8 41 BF 53 00 CE F9 99 5A FA FF DC 62 82 0C AB 13 E4 56 C0 70
    ↪ AD 84 9B BE 37 F7 D0 79 B8 2B 53 41 67 0E FE
[+] SCard response:
89 00

C: 0.090625
[+] SCard request :
80 02 00 00 A0 4E 35 AD DE B0 EC 0A 3A 7F CE F4 41 E2 29 0A 4A 5E BA D4 AD 5B 61 9D FD 26 FD 23
    ↪ C8 8F F7 EE 71 A3 65 65 29 18 94 89 5F B8 19 D2 7C A7 E4 D1 F7 AD 52 DF 18 89 21 09 E9 F8
    ↪  46 13 B6 32 8B CC 18 00 00 00 00 00 00 00 00 00 00 00 00 7F D5 E1 49 A2 32 29 59 BE A5
    ↪ B9 04 23 AF 59 F4 13 87 FA 35 FD 68 9E 74 6A FA 6C 7B 17 09 88 26 4E FA F9 92 A1 AE AE 0A
    ↪  F9 BB 21 86 13 20 8B E5 E8 41 BF 53 00 CE F9 99 5A FA FF DC 62 82 0C AB 13 E4 56 C0 70
    ↪ AD 84 9B BE 37 F7 D0 79 B8 2B 53 41 67 0E FE
[+] SCard response:
80 20 00 00 41 01 00 00 00 00 00 00 00 00 00 00 00 3B 65 C5 85 02 A1 63 4E D0 2F C0 47 5F BC
    ↪ B8 71 7A 1C DE A3 67 50 79 7C 72 BB 5E AA 86 F8 42 60 1A 91 38 A5 DA 06 FE 06 B6 4B 83 24
    ↪  81 3E 5F 59 67 72 05 2F 00 90 00

C: 0.103352
[+] SCard request :
80 20 00 00 41 01 00 00 00 00 00 00 00 00 00 00 00 3B 65 C5 85 02 A1 63 4E D0 2F C0 47 5F BC
    ↪ B8 71 7A 1C DE A3 67 50 79 7C 72 BB 5E AA 86 F8 42 60 1A 91 38 A5 DA 06 FE 06 B6 4B 83 24
    ↪  81 3E 5F 59 67 72 05 2F 00
[+] SCard response:
61 06

[+] SCard request :
00 C0 00 00 06
[+] SCard response:
80 03 01 00 00 10 90 00

V: 1.024481
[+] SCard request :
80 03 01 00 00
[+] SCard response:
80 23 00 00 10 1E B1 6B 64 8A 37 1D 3A 7E 1B 8F 5A 2D 65 9E FA 0B 90 00

C: 0.043752
[+] SCard request :
80 23 00 00 10 1E B1 6B 64 8A 37 1D 3A 7E 1B 8F 5A 2D 65 9E FA 0B
[+] SCard response:
61 0B

[+] SCard request :
00 C0 00 00 0B
[+] SCard response:
00 01 02 03 04 05 06 10 05 07 E6 90 01

V: 0.372549
OK!
```