VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2020 Tadeáš Cvrček

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

KRYPTOGRAFIE NA PLATFORMĚ ARDUINO
CRYPTOGRAPHY ON ARDUINO PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Tadeáš Cvrček**
AUTHOR

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

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Tadeáš Cvrček ***ID:*** 203699
Ročník: 3 ***Akademický rok:*** 2019/20

NÁZEV TÉMATU:

Kryptografie na platformě Arduino

POKYNY PRO VYPRACOVÁNÍ:

Téma práce je zaměřeno na implementaci kryptografických protokolů na platformu Arduino. Student analyzuje dostupné kryptografické knihovny a provede benchmarkové testy kritických algoritmů (modulární operace, operace nad eliptickou křivkou, bilineární párování, hashovací a šifrovací algoritmy atd.) na vybraných zařízeních. Výstupem práce bude funkční implementace přiděleného kryptografického protokolu s ochranou soukromí na platformě Arduino.

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] ARDUINO [online]. 2019Arduino [cit. 2019-09-19]. Dostupné z: https://www.arduino.cc/

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

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

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

ABSTRAKT

Bakalářská práce se věnuje analýze možností využití platformy Arduino pro kryptografické systémy a zároveň implementaci anonymních atributových ověřovacích schémat na této platformě. Obsahem jsou výsledky rešerší dostupných knihoven pro modulární aritmetiku i operace na eliptických křivkách a jejich výkonnostní testy. V návaznosti na výsledky pak byly vybrány vhodné knihovny a provedena praktická implementace moderních anonymních atributových ověřovacích schémat.

KLÍČOVÁ SLOVA

Arduino, modulární aritmetika, operace na eliptických křivkách, kryptografie, kryptografické knihovny, matematické knihovny, kryptografické algoritmy, anonymní atributová ověřovací schémata

ABSTRACT

The bachelor thesis deals with the analysis of possibilities of using Arduino platform for cryptographic systems, while implementing anonymous credential schemes on this platform. The content includes the results of searches of available libraries for modular arithmetic and operations on elliptic curves and their performance tests. Following the results, suitable libraries were selected and a practical implementation of modern anonymous credential schemes was performed.

KEYWORDS

Arduino, modular arithmetics, elliptic-curve operations, cryptography, cryptographic libraries, mathematic libraries, cryptographic algorithms, anonymous credential schemes

CVRČEK, Tadeáš. *Kryptografie na platformě Arduino*. Brno, 2020, 62 s. Bakalářská práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce: Ing. Petr Dzurenda, Ph.D.

PROHLÁŠENÍ

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

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

Brno ....

podpis autora

PODĚKOVÁNÍ

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

Tato práce vznikla jako součást klíčové aktivity KA6 - Individuální výuka a zapojení studentů bakalářských a magisterských studijních programů do výzkumu v rámci projektu OP VVV Vytvoření double-degree doktorského studijního programu Elektronika a informační technologie a vytvoření doktorského studijního programu Informační bezpečnost, reg. č. CZ.02.2.69/0.0/0.0/16_018/0002575.

EVROPSKÁ UNIE
Evropské strukturální a investiční fondy
Operační program Výzkum, vývoj a vzdělávání

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

Projekt je spolufinancován Evropskou unií.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

1 Úvod

V poslední době začalo sílit šíření IoT (*Internet of Things*) zařízení a tendence o rozšíření možností komunikace prostřednictvím sítě internet, za použití chytrých telefonů nebo počítačů, s věcmi, které používáme na denní bázi. Může to být kávovar, může to být regulace topného systému. Součástí věci připojené do IoT musí být zařízení, síťové rozhraní, zprostředkovávající zabezpečenou komunikaci přes síť. Právě při implementaci kryptografických protokolů se ale naráží na výkonnostní bariéry, jež plynou z charakteru použitých komponent. Není možné používat plnohodnotné počítače v IoT, jelikož by to nebylo výhodné z vícero důvodů Jednak by se takový produkt výrazně prodražil a měl by vyšší provozní náklady, ale také by byl zbytečně složitější. Taky proto se pro takové účely používají zařízení s omezeným výkonem.

Příchod trendu *Průmysl 4.0* [2] přináší právě tendenci o zvýšení integrace věcí z reálného světa do kybernetického prostoru. To sebou nese otázky zabezpečení provozu před nechtěným únikem provozních dat IoT nebo osobních údajů. *Průmysl 4.0* by měl využívat i systém umělé inteligence pro zabezpečení procesů, které byly předtím vykonávány zaměstnanci. O to důležitější je reálné zabezpečení před kybernetickými hrozbami, jelikož útočník, který by kompromitoval takový systém, by mohl napáchat nemalé finanční i materiální škody.

Tato bakalářská práce se věnuje analýze možností využití zařízení platformy Arduino [3] pro implementaci kryptografického systému a cíleně prozkoumat možnosti veřejně dostupných knihoven a otestovat je. Důležité je provést benchmarkové testy výpočtů modulární aritmetiky a operací nad eliptickými křivkami. Je možné ale brát i v potaz výkonnostní testy jiných kryptografických algoritmů.

Vyhodnocením testů, tedy i díky výstupu práce, lze následně získat ucelenou představu o tom, čeho jsou tato zařízení schopna, jakým způsobem je možné je využít v následné implementaci a kde jsou jejich hranice, popřípadě jaké knihovny jsou pro tato zařízení k dispozici a co od nich lze očekávat.

V praktické části je následně provedena implementace moderních atributových schémat využívaných v systémech řízení přístupu. Tato schémata využívají operace na eliptických křivkách i modulární aritmetiku.

2 Vestavěné systémy a kryptografie

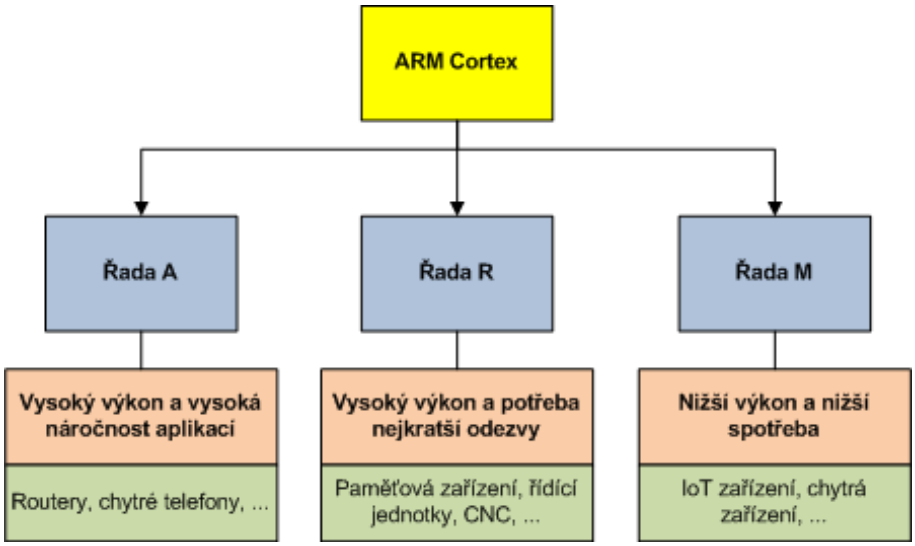
Vestavěné systémy, z angl. *embedded systems*, jsou systémy založené na zařízeních s nízkým výpočetním výkonem, a tedy i spotřebou. Pro takové systémy se používají procesory založené na architekturách ARM (*Advanced RISC Machines*) a AVR. ARM představuje rodinu 32 bitových procesorů, zatímco AVR procesory jsou výhradně 8 bitové, nicméně existují i 32 bitové modely.

2.1 Architektura ARM

Společnost ARM založily v roce 1990 firmy Apple, VLSI Technology a Acorn. Cílem bylo vyrábět procesory pro zařízení Apple. na přelomu 20. a 21. století již firma pronikla do zařízení různých značek a lze ji od té doby považovat za globální. Prvním mobilním telefonem postaveným na platformě ARM pak byla Nokia 6110.

Nyní se společnost ARM zaměřuje především na rodinu procesorů *Cortex* [4] [5], viz obrázek 2.1. Ta zastřešuje řady označované jednotlivými písmeny názvu společnosti a odlišují je především cílená prostředí. Existují tři základní rozřazení:

- **modelová řada A** – zaměřena na poskytnutí vysokého výkonu,
- **modelová řada R** – specializuje se na vysoký výkon, ale zároveň minimalizuje odezvy (například řídící jednotky),
- **modelová řada M** – specializace na segment zařízení s nízkou spotřebou (například IoT zařízení).



Obr. 2.1: Diagram rodiny procesorů ARM Cortex

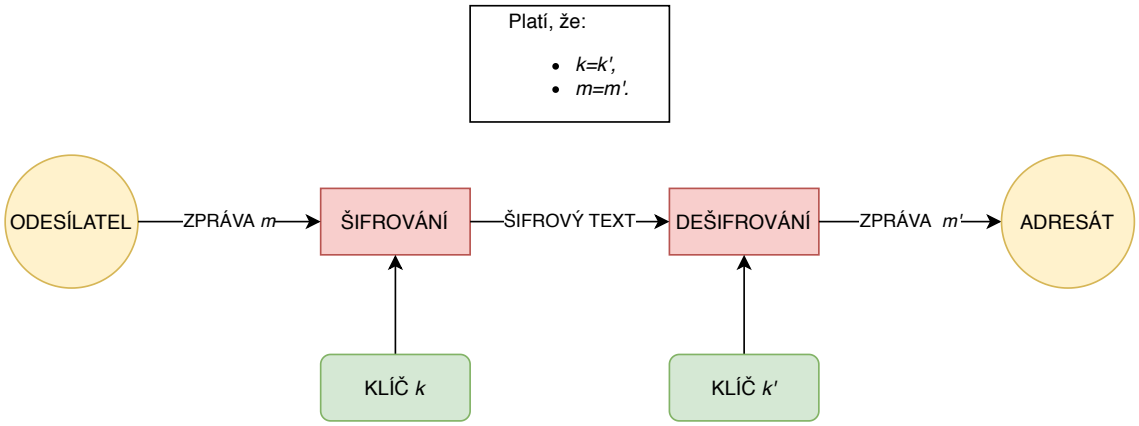
2.2 Architektura AVR

Architekturu AVR [6] vytvořila americká firma Atmel, která byla založena v roce 1984. Jejich prvním cílem bylo umožnění ukládání dat v režimu NVM (*Non-Volatile Memory*) do paměti, která je fyzicky součástí kontroléru. NVM vyjadřuje schopnost paměti udržet data v paměti i při odpojení od zdroje napětí. Firma byla jako první úspěšná při integraci NVM EEPROM (*Electrically Erasable Programmable Read-Only Memory*) paměti do kontroléru.

AVR kontroléry lze najít především v zařízeních s omezeným výkonem. Praktické uplatnění bylo nalezeno i v automobilovém průmyslu (například u systémů kontroly tlaku v pneumatikách, systémů ochrany před odcizením vozidla, ovladačů airbagů apod.). na architektuře AVR vznikla též platforma Arduino.

2.3 Kryptografická primitiva a protokoly

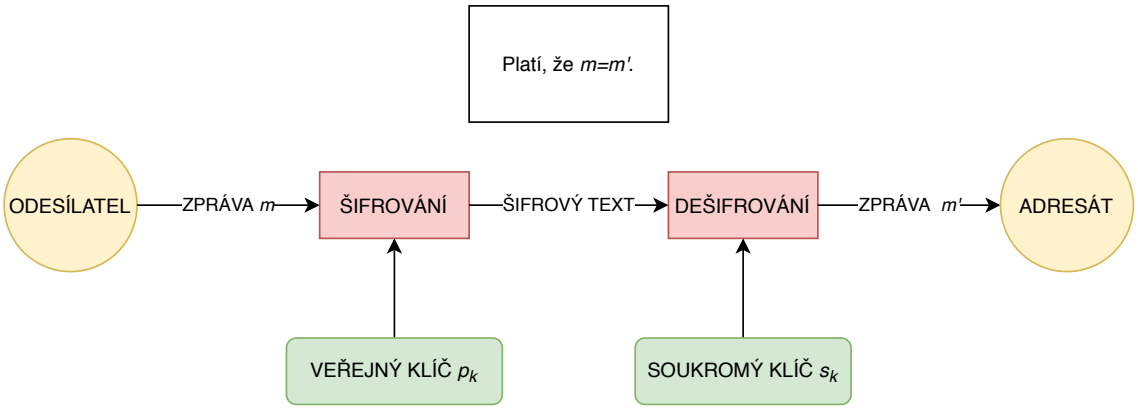
Existují dva typy kryptografie: **symetrická** a **asymetrická**. Liší se v přístupu k šifrování, resp. dešifrování, co se použitého klíče týče [7]. Symetrická kryptografie, znázorněná na obrázku 2.2, používá na šifrování i dešifrování stejný klíč, který má délku danou vybraným algoritmem. Příkladem může být zastaralý DES (*Data Encryption Standard*) či novější bezpečný AES (*Advanced Encryption Standard*).



Obr. 2.2: Znázornění principu symetrické kryptografie

Asymetrická kryptografie, znázorněná na obrázku 2.3, je založena na nemožnosti řešit určité matematické problémy. Využívají se operace na konečných polích a operace na eliptických křivkách. Jedná se o kryptografii, kde se využívají dva klíče odlišné pro šifrování i dešifrování, které jsou obvykle označovány jako veřejný a soukromý. Mezi asymetrické algoritmy patří například RSA (název dle iniciál Rivest, Shamir, Adleman), což je algoritmus vhodný pro šifrování i práci s digitálními

podpisy, nebo algoritmy využívající eliptické křivky, např. ECDH (*elliptic curve Diffie–Hellman*) a ECDSA (*elliptic curve digital signature algorithm*).



Obr. 2.3: Znázornění principu asymetrické kryptografie

Dalšími pojmy, které se v kryptografii používají jsou:

- **autentizace** – ověření pravosti identity,
- **autorizace** – povolení entitě vstoupit do systému,
- **řízení přístupu** – identifikace pravomocí entity.

Moderní atributové autentizaci, využívající eliptických křivek, je věnována sekce Atributová autentizace dále.

2.3.1 Modulární aritmetika

Modulární aritmetika [8] se opírá o matematickou operaci modulo, na jejímž výstupu je zbytek po dělení. Zavádí se pojem kongruence (symbol „$\equiv$“ – tj. dvě různá čísla mají po dělení stejným číslem stejný zbytek (například $9 \equiv 33 \bmod 4$).

Všechna čísla, která jsou kongruentní v modulo a, jsou obsažena v tzv. kongruentní (či zbytkové) třídě. Příkladem může být množina $\{..., 2, 9, 16, ...\}$ v modulo 7. Toto je výhodné pro využití v kryptografii, neboť je-li k dispozici výstup z operace modulo, není jednoduché zjistit, které číslo bylo konkrétně na vstupu. Vlastnosti sčítání, odčítání a násobení v modulární aritmetice lze vyjádřit rovnicemi 2.1, 2.2 a 2.3.

$$(a + b) \bmod c = (a \bmod c + b \bmod c) \bmod c \tag{2.1}$$

$$(a - b) \bmod c = (a \bmod c - b \bmod c) \bmod c \tag{2.2}$$

$$(a \cdot b) \bmod c = (a \bmod c \cdot b \bmod c) \bmod c \tag{2.3}$$

2.3.2 Kryptografie nad eliptickými křivkami

Moderní kryptografické algoritmy, jako třeba ECDH nebo ECDSA, jsou založeny na výpočtech na eliptických křivkách a výpočtech modulární aritmetiky. Jakákoliv eliptická křivka [9] je definovatelná pomocí obecné Weierstrassovy rovnice 2.4.

$$y^2 = ax^3 + bx^2 + cx + d \tag{2.4}$$

Je ale i možné provést transformaci, jejímž výsledkem je zjednodušená Weierstrassova rovnice [10] eliptické křivky nad prvočíselným polem $\mathbb{F}_p$. Zjednodušená Weierstrassova rovnice je zapsána v rovnici 2.5.

$$E/\mathbb{F}_p : y^2 = x^3 + ax + b \tag{2.5}$$

Mezi koncovými body komunikace jsou známy určité parametry (tzv. doménové parametry) eliptické křivky – prvočíslo p (definice tělesa); konstanty a, b (dle rovnice definující používanou křivku); generující bod křivky, který se na křivce nachází; řád generujícího bodu n udávající počet bodů v rámci vygenerované grupy a kofaktor h.

Výhodou eliptických křivek je jejich rychlost, která je dána kratší délkou kryptografických klíčů oproti standardním kryptosystémům založených na faktorizaci velkých celých čísel (například RSA) a problému diskrétního logaritmu (například DSA). Pro práci s křivkami nám stačí menší čísla než u algoritmů, které je nevyužívají, ale bezpečnostní úroveň zůstává stejná.

Sčítání bodů na eliptické křivce

Obecně lze sčítání bodů na eliptické křivce vyjádřit rovnicí 2.6.

$$F_1 + F_2 = F_3 \tag{2.6}$$

V případě sčítání bodů mohou vzniknout dvě situace:

1. $F_1 = F_2$ – body, které mají být sečteny jsou totožné; jedná se o zdvojení bodu (násobení bodu dvěma),
2. $F_1 \neq F_2$ – body, které mají být sečteny jsou rozdílné.

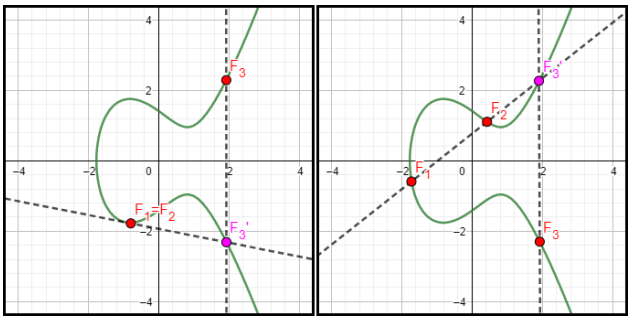
Zdvojení bodu – pakliže platí rovnost $F_1 = F_2$, operace zdvojení na eliptické křivce se provádí následovně (graficky znázorněno na obrázku 2.4 vpravo):

1. je vedena tečna ke křivce bodem $F_1 = F_2$,
2. průsečík tečny s eliptickou křivkou lze označit jako bod $F_3^{'}$,
3. osovou souměrností bodu $F_3^{'}$ přes osu X vzniká bod F_3.

Lze tedy tvrdit, že sčítání totožných bodů je stejné, jako násobení jednoho z bodů číslem 2 (zdvojení bodu). To znamená, že platí rovnost $F_3 = F_1 + F_1 = 2F_1$.

Sčítání různých bodů – pakliže platí, že $F_1 \neq F_2$, operace sčítání na eliptické křivce se provádí následovně (graficky znázorněno na obrázku 2.4 vlevo):

1. je vedena přímka body F_1 a F_2,
2. průsečík přímky s eliptickou křivkou lze označit jako bod F_3',
3. osovou souměrností bodu F_3' přes osu X vzniká bod F_3.



Obr. 2.4: Grafické znázornění zdvojení bodů a sčítání bodů

Při sčítání dvou různých bodů platí rovnost $F_3 = F_1 + F_2$.

Násobení bodu na eliptické křivce skalárem

Násobení bodu ležícího na eliptické křivce skalárem je možné si představit pomocí ilustrativního postupu pro násobení číslem 4 (znázorněno na obrázku 2.5):

1. existuje bod G ležící na eliptické křivce,
2. tečna ke křivce procházející bodem G protíná křivku v bodě $2G'$,
3. osovou souměrností bodu $2G'$ přes osu X vzniká bod $2G$,
4. bod $3G$ by se získal vytvořením přímky body G a $2G$, nalezením průsečíku s křivkou a osovou souměrností přes osu X,
5. stejným způsobem by se provedla ještě čtvrtá iterace, na jejímž konci by byl bod $4G$,

Násobení bodu je dáno jeho sčítáním, a tedy platí rovnice 2.7.

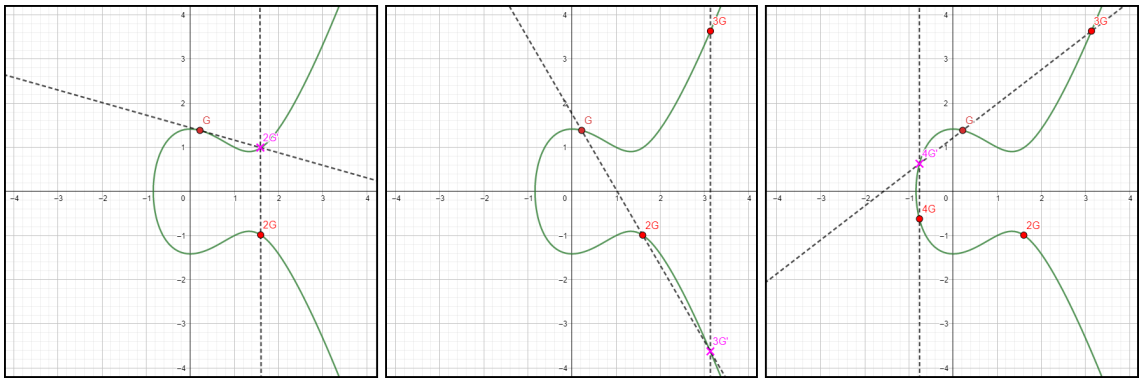
$$4G = G + G + G + G \tag{2.7}$$

Na obrázku 2.5 byla použita technika zdvojení bodu (*Point Doubling*) v kombinaci se sčítáním bodů, jak je znázorněno v rovnici 2.8.

$$4G = 2G + G + G \tag{2.8}$$

V praxi by ale bylo použito jedno zdvojení bodu a jedno sečtení, což je zapsáno rovnicemi 2.9 a 2.10.

$$H = 2G \tag{2.9}$$



Obr. 2.5: Grafické znázornění násobení bodu eliptické křivky skalárem

$$4G = H + H \tag{2.10}$$

Důležité je, že existuje-li bod aG, kde a představuje skalár, kterým je násoben bod G, není pro útočníka v rozumném čase možné skalár zjistit, protože by to znamenalo vyřešení problému diskrétního logaritmu na eliptické křivce. Implementace operace násobení bodu skalárem pak probíhá pomocí běžných matematických postupů, jako je třeba využití rovnice tečny ke křivce apod. z toho plyne, že není nutné používat specializované výpočetní jednotky (koprocesory), nicméně je to žádoucí, jelikož se snižují nároky na hlavní procesor, jelikož v takovém případě se jedná o implementaci hardwarovou, ne softwarovou, a proto by byly i výpočty rychlejší.

2.4 Atributová autentizace

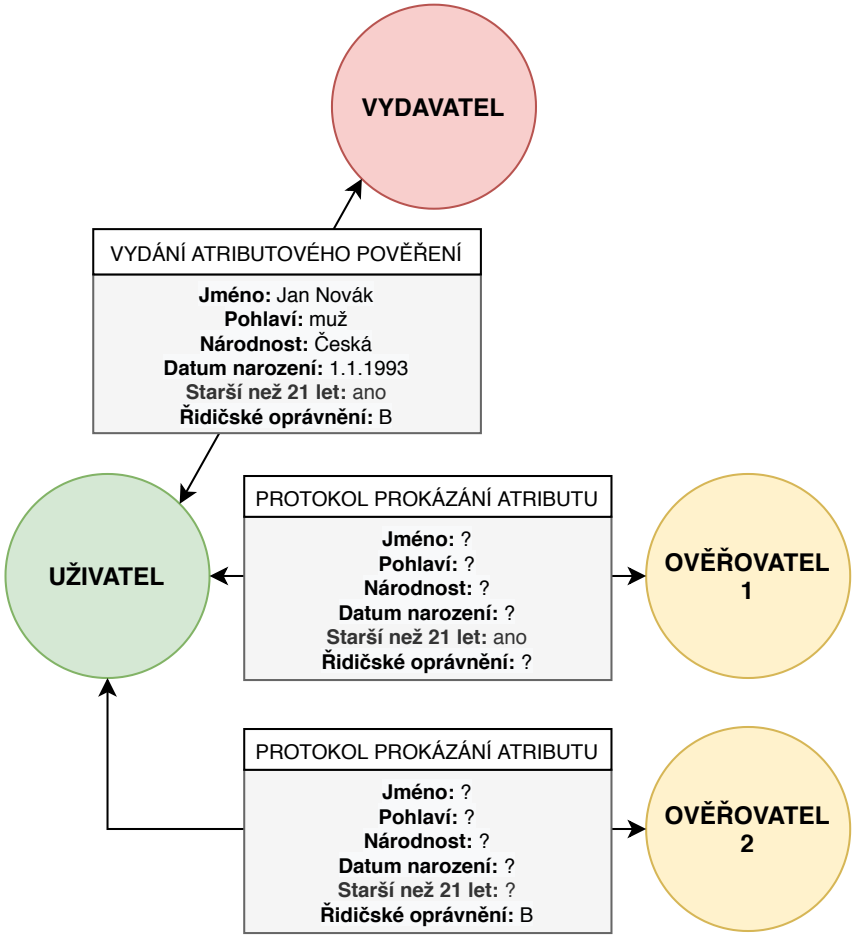
Tato kapitola se věnuje praktické implementaci protokolu RKVAC navrženého v rámci projektu Právní a technické prostředky pro ochranu soukromí v kyberprostoru, r.č. TL02000398 a vychází z publikací [11] a [12].

Autentizace označuje ověření identity uživatele. Klasicky je autentizace založena na ověření pomocí ID uživatele, následně uživatel musí dokázat, že je tím, za koho se vydává. K tomu může použít tajné heslo, svoji čipovou kartu nebo biometrické vlastnosti, jako je například otisk prstu.

Atributová autentizace ale není, na rozdíl od klasické autentizace, postavena na kontrole identit uživatelů, místo toho ale dochází k ověřování disponibility konkrétního atributu, tj. vlastnosti uživatele. Atributy mohou být například věk, zbrojní průkaz osoby, pohlaví, národnost apod.

Proces ověřování je založen na faktu, že není nezbytné uvolňovat další soukromé informace. Pakliže je úspěšně prokázána disponibilita, uživateli je umožněn přístup do systému nebo k chráněným službám, přičemž zůstává v anonymitě. Jako ochranný

mechanizmus zde vystupuje revokační autorita, která má schopnost uživatele vyloučit ze systému v případě, že je to potřeba (například porušení pravidel nebo odhalení identity).



Obr. 2.6: Schéma atributové autentizace

Schémata využívající atributové pověření ABC (*Attribute-Based Credential schemes*) se skládají z těchto základních entit, jak je možné vidět na schématu 2.6:

- **uživatel**: entita, která prokazuje nějaké vlastnosti,
- **ověřovatel**: entita, která ověřuje platnost předaných uživatelských atributů,
- **veřejná autorita**: entita, která spravuje uživatele (může zahrnovat vydavatele a revokační autoritu).

Základní fáze těchto protokolů jsou:

- **nastavení**: nastavuje se prostředí a pravidla,
- **vydávání**: registrace nového uživatele (uživatel získá kryptograficky zabezpečené atributy),
- **prokazování a ověřování**: ověřovací entita ověřuje uživatelův důkaz držení atributů,

- **revokace**: revokační autorita má právo vyřadit uživatele ze systému (dojde k zneplatnění atributů).

KVAC schéma (*Keyed-Verification Anonymous Credential scheme*) se od ABC schéma liší tím, že ověřovatel potřebuje znát tajné klíče vydavatele a jedná se tedy o stejnou entitu, jako vydavatel. RKVAC (*Revocable Keyed-Verification Anonymous Credentials*) schéma pak rozšiřuje KVAC schéma o přítomnost revokační autority.

3 Platforma Arduino a kryptografická podpora

Platforma Arduino je postavena na jednočipových počítačích, které jsou, společně s dodávaným software, který tvoří IDE (*Integrated Development Environment*) a bootloader, vyvíjeny v duchu open-source projektu. Pomyslným srdcem těchto zařízení jsou kontroléry ATmega od společnosti Atmel. Výjimku tvoří model DUE, který využívá procesorové architektury ARM. K desce lze připojit řadu periferních zařízení, jako jsou například senzory, servomotory, LED diody, síťová rozhraní a podobně. Vysoký počet možností, jak lze s zařízením nakládat, z něj vytvořil jednu z nejpopulárnější elektronických stavebnic na trhu.

Arduino je open-source projekt a na vytvoření klonu není nezákonný čin, ale nesmí se použít značky Arduino, protože to už by zasáhlo do práv společnosti [13]. v e-shopech lze běžně nalézt klony, které jsou značně levnější, přičemž funkcionalita zůstává zachována na úrovni originálu. Kvalitu zpracování ale mnohdy nelze s originálem srovnávat – příkladem může být použití i co nejlevnějších klonů jednotlivých čipů, kvalitou komponent i úroveň pájecích prací. Existují tedy klony [14] označované jako FreeDuino nebo Arduelo.

Alternativou pro IoT mohou být produkty společnosti Google, která umožňuje nákup kitů pro vytváření prototypů a vyvíjí Android Things, což je operační systém pro vestavěné systémy. Konkurenční společnost Microsoft žádné takové řešení nenabízí, zaměřuje se ale na podporu cloud API (*Application Programming Interface*), které je součástí platformy Microsoft Azure.

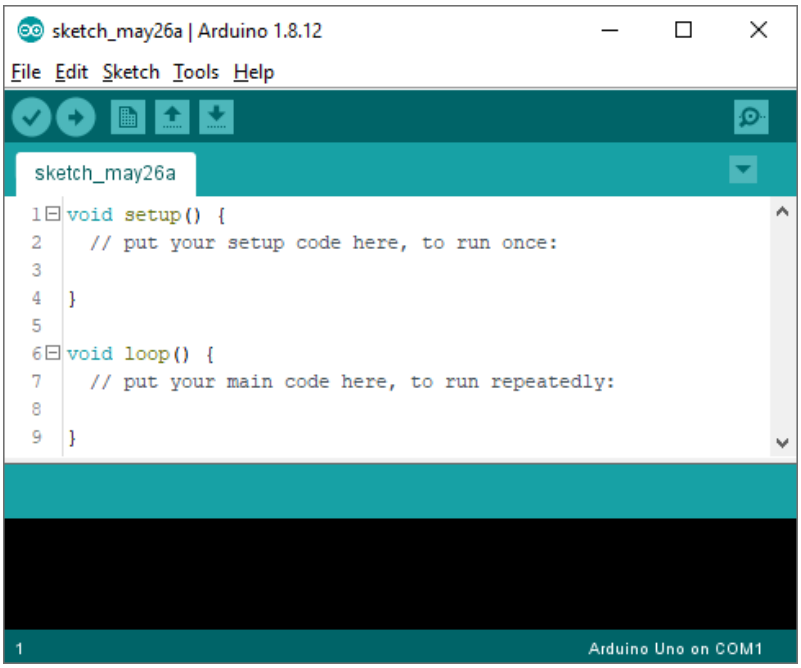
3.1 Software

Aplikace se vyvíjí nejčastěji pomocí zdarma dostupného rozhraní Arduino IDE od výrobce v jazyce C++. Lze použít i komerční aplikace, mezi které patří například Microsoft Visual Studio od verze 2017. Projekt je následně zkompilován pro příslušný model a jeho revizi. Po kompilaci proběhne nahrání přes sériovou linku, která je zprostředkovaná přes rozhraní USB, do flash paměti zařízení.

Základem všech programů platformy Arduino jsou dvě metody (viz obrázek 3.1):

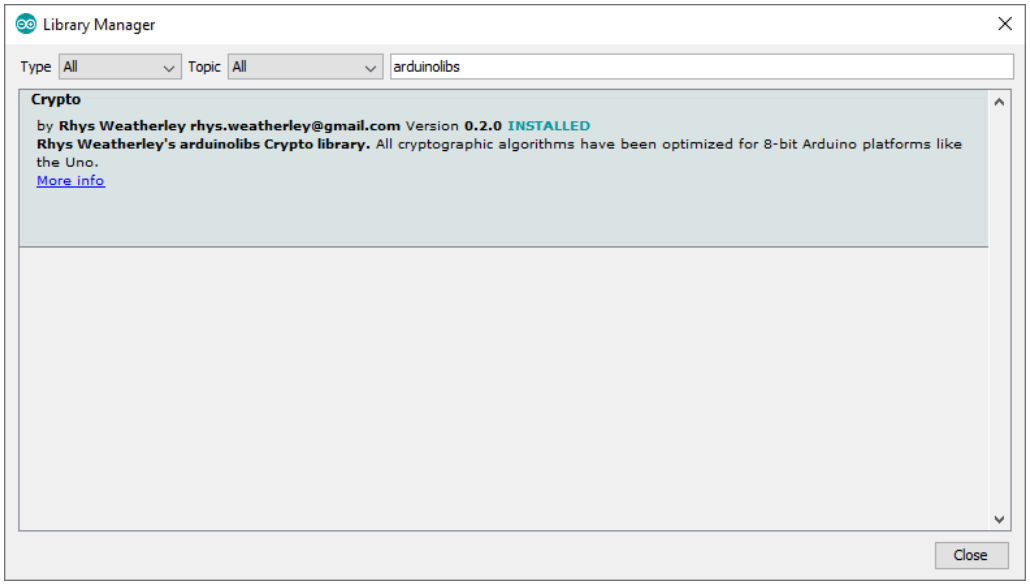
1. `void setup()` - metoda, která se spustí jako první v dané relaci a slouží především k inicializaci zařízení
2. `void loop()` - metoda, která *„běží donekonečna“* a navazuje na `void setup()` a obsahuje hlavní kód programu

Kromě vlastního kódu, lze v rámci projektů použít i knihovny třetích stran, jimiž je možné urychlit vývoj. Tyto knihovny jsou k dispozici na internetu většinou ve formě



Obr. 3.1: Nový projekt v Arduino IDE

připraveného archivu *.zip*, a také prostřednictvím repositáře knihoven platformy Arduino, který je integrován přímo ve vývojovém prostředí, viz obrázek 3.2. Repositář stále neobsahuje veškeré dostupné knihovny, avšak na rozdíl od GitHubu nebo jiných zdrojů, zde je jistota stabilního provozu zařízení při jejich využití. Distribuce knihoven se nikdy neprovádí v již zkompilovaném stavu, jako například u *statických knihoven C++*, ale ve formě zdrojového kódu s přiloženým souborem s metadaty.



Obr. 3.2: Repozitář knihoven v Arduino IDE

Knihovny umožňují také modularitu kódu (jsou tak vyvíjeny), jelikož u zařízení s nižším výkonem není žádoucí, aby existoval kód, který je neefektivní nebo který dokonce ani není využíván (tzv. *"mrtvá větev programu"*), protože během překladu dochází k analýze kódu a vyhodnocení, které části jsou skutečně potřeba k běhu. Tedy základem programu pro Arduino bývá C++ zdrojový kód se dvěmi základními metodami (viz výše), ze kterého se odkazuje na knihovny. Překladač kódu dokonce provádí takové optimalizace, že do výsledného sestavení se nedostane kód, který by nebyl využit – zde záleží ale také na vývojářské optimalizaci.

Níže uvedený kód, viz výpis 3.1, představuje situaci, při které by kompilátor ignoroval existenci funkce `void printWorld()`. Výstupem v sériovém monitoru by byl text *„Hello “* (bez uvozovek), protože z metody `void loop()` je volána pouze funkce výpisu slova *Hello* a následně je ještě vypsán symbol mezery. Metoda `void setup()` v tomto případě slouží pouze k navázaní komunikace přes sériovou linku.

Výpis 3.1: Příklad kódu pro Arduino

```
1  void setup() {
2    Serial.begin(9600);
3  }
4
5  void loop() {
6    printHello();
7    Serial.print(" ");
8  }
9
10 void printHello()
11 {
12   Serial.print("Hello");
13 }
14
15 void printWorld()
16 {
17   Serial.print("World");
18 }
```

3.2 Hardware

Modelů jednočipových desek platformy Arduino je vícero [15]. Nejznámější je asi model UNO, dále pak NANO a MEGA (popř. k MEGA alternativní DUE, které používá kontrolér architektury ARM). Modely se liší použitým kontrolérem, velikostí paměti flash i SRAM, fyzickou velikostí, konektivitou a podobně. Podrobněji lze rozdíly vidět v tabulce 3.1.

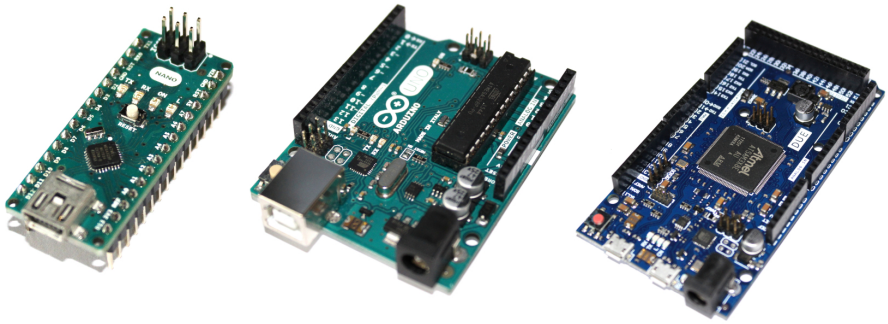
Tab. 3.1: Srovnávací tabulka vybraných modelů platformy Arduino

Model	Kontrolér	Takt (MHz)	Piny (A/D)	Flash (kB)	EEPROM (kB)	SRAM (kB)
NANO (AVR)	ATmega328	16	8/14	32	1	2
UNO (AVR)	ATmega328	16	8/14	32	1	2
MEGA (AVR)	ATmega2560	16	16/54	256	4	8
DUE (ARM)	ATSAM3X8E	84	14/54	512	–	96

Konektivitu zajišťují analogové a digitální piny. Analogové piny je vhodné použít například k měření neelektrických veličin, digitální zase pro případnou komunikaci, buď s uživatelem, nebo například s dalším zařízením.

K ukládání programů slouží flash paměť. Jedna buňka této paměti má zvládnout až deset tisíc zápisů. Pokud chceme trvale ukládat určité hodnoty, máme k dispozici paměť EEPROM. Žádný z modelů neobsahuje grafický akcelerátor, matematický ani kryptografický koprocesor. Veškeré operace musí provádět kontrolér.

Zatímco Arduino UNO (na obrázku 3.3 uprostřed) představuje asi nejlepší kombinaci ceny a výkonu a je nejčastější volbou pro seznámení s platformou (jedná se o součást sady *Arduino Starter Kit*), tak NANO sází na miniaturní rozměry a MEGA zase na výkon a vyšší počet konektorů. Alternativou k modelu MEGA je, díky jiné architektuře výkonnější, DUE (na obrázku 3.3 vpravo).



Obr. 3.3: Arduino NANO (vlevo), Arduino UNO (uprostřed) a Arduino DUE (vpravo)

Model DUE používá jako kontrolér procesor architektury ARM. Jedná se o 32 bitový procesor, který je oproti všem 8 bitovým kontrolérům na platformě o poznání rychlejší a nabízí i více paměti. Pokud by bylo uvažováno o využití platformy Arduino pro vestavěné systém, model DUE by byl pravděpodobně jasnou volbou, pokud by splňoval parametry důležité pro daný projekt (například rozměry nebo poměr ceny a výkonu).

Kromě výkonnostních a paměťových rozdílů, má využití procesoru ARM další dopady. v prvé řadě je to přítomnost dvou USB portů ve variantě microUSB –

jeden slouží pro komunikaci přímo s procesorem, druhý lze použít na programování a vývoj (spojení s procesorem zprostředkovává pomocí čipu ATmega16U2). Byť oba porty lze použít k nahrání programu na desku, druhý je výrobcem preferovaný. Dalším rozdílem oproti architektuře AVR kontrolérů ATmega je napětí na pinech. To totiž není $5V$, ale $3,3V$. na to je třeba při návrhu a výrobě jakéhokoliv projektu pamatovat, neboť napětí vyšší než $3,3V$ může procesor nenávratně zničit [16].

3.3 Dostupné kryptografické knihovny

Pro aplikaci kryptografických protokolů je potřeba, aby bylo možné využívat operaci *modulo*. v případě programovacího jazyka C++ na modelu Arduino UNO se jedná o operátor „*%*“, který vrací zbytek po dělení. Takový operátor je ale validní (v případě celých čísel) pouze pro datové typy „*long*“ (32 bitů), „*int*“ (16 bitů) a „*short*“ (16 bitů) a jejich deriváty („*unsigned long*“, „*unsigned int*“, „*unsigned short*“). Přehled celočíselých datových typů je uveden v tabulce 3.2 [17].

Tab. 3.2: Celočíselné datové typy (model Arduino UNO)

Datový tip	Počet bitů	Minimum	Maximum
short	16	−32 768	32 768
int	16	−32 768	32 768
long	32	−2 147 483 648	2 147 483 648
unsigned short	16	0	65 535
unsigned int	16	0	65 535
unsigned long	32	0	4 294 967 295

Je možné si povšimnout, že mezi typy „*short*“ a „*int*“ není u modelu UNO žádný rozdíl. Běžně bývá v informatice „*int*“ dvakrát větší (totéž platí o vztahu „*int*“ a „*long*“), avšak zde bylo nutné uměle omezit vlastnosti programovacího jazyka, aby vyhovoval dostupným zdrojům. Pokud je potřeba, aby bitové velikosti datových typů odpovídaly standardu *C++17*, je nutné použít zařízení Arduino DUE.

Knihovna, která by byla použitá v kryptografickém systému, by musela být schopna pracovat s mnohem většími čísly . v případě využití eliptických křivek se jedná o 160, 192, 224, 256, 384 a 512 bitů, avšak bez eliptických křivek by bylo potřeba podpory ještě větších čísel (například pro RSA 2048 bitů). Volně dostupné knihovny, které podporují práci s velkými celými čísly jsou *BigNumber* [18] a *arduinolibs* [19]. Obě zmíněné knihovny lze dohledat na GitHubu. Předpokladem využití knihoven k implementaci kryptografického protokolu je, aby podporovaly tyto operace:

- modulární redukci (tj. $a \bmod c$),
- modulární násobení (tj. $(a \cdot b) \bmod c$),
- modulární mocnění: (tj. $(a^b) \bmod c$).

Nejvýznamnější knihovnou pro práci s eliptickými křivkami je knihovna *micro-ecc*, která je k dispozici na GitHubu pod licencí *BSD 2-Clause* [20]. Tato knihovna má implementované algoritmy ECDH a ECDSA spolu s řadou eliptických křivek jako takových. Návrh byl mířen přímo na zařízení s omezeným výkonem – dle autora má kód po kompilaci pro ARM Cortex M0 velikost 3 kB, a proto se jedná v tomto ohledu o ideální knihovnu. Lze také ocenit odolnost vůči útokům postranními kanály (autor se nezmiňuje o odolnosti za různých podmínek, mezi které patří využití paměti apod.) a podporu pro 8, 32 i 64 bitové architektury, což teoreticky znamená, že knihovnu lze bez úprav použít na zařízeních Arduino UNO, Raspberry Pi i počítači současně. Zdrojový kód je napsaný převážně v jazyce C s minimálním využitím jazyka symbolických adres [21].

3.4 Výkonnostní testy

Veškeré testy zaznamenané v této práci byly prováděny na modelech NANO, UNO a DUE. Zařízení bylo napájeno z počítače pomocí sběrnice USB ve verzi 2.0 a přes komunikační kanál na sběrnici také probíhala komunikace. K monitorování byl použit integrovaný nástroj v Arduino IDE pro práci se sériovou sběrnicí, popř. nástroj Putty, který je volně ke stažení na internetu (na rozdíl od Arduino IDE umožňuje kopírovat standardní výstup).

Vzhledem k charakteru výstupního sestavení programů, lze měření provádět pouze jediným postupem: **zaznamenání počátečního času** (tj. čas před měřenou operací), **zaznamenání koncového času**, **výpočet rozdílu zaznamenaných časů** (přesnost na milisekundy).

U měření nelze používat *breakpointy*, tj. místa, kde se kód zastaví a je možné zahájit diagnostiku, jelikož strojový kód Arduina neumožňuje „*promítnutí*“ na kód zdrojový. Dochází tedy ke kompilaci přímo do strojového jazyka a žádný používaný čip ladící procedury nepodporuje, a proto nelze využívat klasických nástrojů známých z vývoje aplikací s využitím vysokoúrovňových programovacích jazyků (například C# nebo Java).

Další faktory, které se týkají pozorování jsou množství využité flash paměti a operační paměti. Nedostatek operační paměti může vést k nestabilnímu chování zařízení nebo dokonce úplné nefunkčnosti.

3.4.1 Výpočty modulární aritmetiky

K testu modulární redukce (knihovna *BigNumber*) bylo použito šest měření, ve kterých bylo číslo o velikosti a bitů redukováno jiným číslem též o velikosti a bitů. Test používal bitové délky z množiny $16, 32, 64, 128, 256, 512$. o úspěšnosti výpočtu se rozhodovalo porovnáním výstupních hodnot z kontrolní aplikace na stolním počítači, která byla pro tyto účely napsána v programovacím jazyce C#. Modulární násobení bylo realizováno v šesti měřeních přes vzorec $(a \cdot b) \bmod n$, kde a, b i n jsou čísla nerovnající se, avšak stejně veliká (16, 32, 64, 128, 256 a 512 bitů). Po výpočtu byl výsledek opět zkontrolován pomocí počítače. Měření výkonu modulárního mocnění bylo dosaženo znovu pomocí šesti měření a vzorce $(a^b) \bmod n$, kde a, b i n jsou čísla nerovnající se, avšak stejně veliká (16, 32, 64, 128, 256 a 512 bitů).

Knihovna *BigNumber* byla naprogramována *Nickem Gammonem*. Stále ji jako jediný programátor spravuje a vyvíjí. Dle oficiálních zdrojů [18] podporuje čísla o velikosti až 256 bitů. Zkoumání zdrojového kódu odhalilo, že knihovna používá pole datového typu „*byte*“ s následnou implementací (i optimalizací) operací nad takovými čísly. Takovým způsobem programátor obešel omezující vlastnosti primitivních datových typů pro celá čísla.

Podporuje sčítání, odčítání, násobení, dělení, mocnění, modulární redukci a modulární mocnění. Knihovna sice nepodporuje modulární násobení, avšak toho lze dosáhnout poměrně jednoduše s využitím vlastností modulární aritmetiky, do kterých patří vztah $(a \cdot b) \bmod c = (a \bmod c) \cdot (b \bmod c) \bmod c$.

Nespornou výhodou této knihovny je její velikost a jednoduchost použití jejích funkcí. Při zkompilovaném stavu všech testů byla paměť zařízení využita přibližně z 20%, což ve výsledku znamená, že je ideální jako jakýsi „*základní kámen*“ pro implementaci rozsáhlejšího algoritmu. Pokud knihovna přesáhla během měření své limity, program přestal pracovat.

Měření na architektuře AVR

Knihovna nebyla schopna zpracovat číslo o velikosti 512 bitů ($a = 2^{512} - 1$). Rychlost redukce byla ale velice dobrá a nepřesáhla 9 ms (viz příslušná část tabulky 3.3), což lze považovat za úctyhodný výkon, s ohledem na kapacity jaké má kontrolér Arduina k dispozici. u malých čísel, 8 bitů a 16 bitů, byl dokonce zaznamenán čas 1 ms.

Při modulárním násobení je možné u této knihovny použít nejvýše 256 bitová čísla, viz příslušná část tabulky 3.3. Tento test odhalil, že zřejmě je možné pracovat s čísly většími než 256 bitů, avšak za předpokladu, že je neukládáme do proměnné datového typu knihovny. Je tedy možné mít na vstup nějaké metody knihovny číslo větší než 256 bitů, ale uložit jej již není stabilní proces. Jakmile knihovna měla

Tab. 3.3: Výsledky modulárních operací pro architekturu AVR (knihovna *BigNumber*)

Modulární redukce						
Zařízení	16 bitů	32 bitů	64 bitů	128 bitů	256 bitů	512 bitů
NANO	1 ms	2 ms	4 ms	6 ms	9 ms	nelze
UNO	1 ms	2 ms	3 ms	5 ms	9 ms	nelze
Modulární násobení						
Zařízení	16 bitů	32 bitů	64 bitů	128 bitů	256 bitů	512 bitů
NANO	1 ms	5 ms	12 ms	35 ms	119 ms	nelze
UNO	1 ms	5 ms	11 ms	35 ms	120 ms	nelze
Modulární mocnění						
Zařízení	16 bitů	32 bitů	64 bitů	128 bitů	256 bitů	512 bitů
NANO	46 ms	211 ms	1172 ms	7632 ms	nelze	nelze
UNO	45 ms	214 ms	1170 ms	7627 ms	nelze	nelze

za úkol pracovat s 512 bitů velkým číslem, které ještě bylo umocněno dalším číslem o velikosti 512 bitů, přestal program pracovat.

Nízké hodnoty při modulárním mocnění knihovna v testu zvládala poměrně dobře. u větších čísel, tj. 128 bitů a více, ovšem výsledky nejsou příliš přívětivé, viz příslušná část tabulky 3.5. Kryptografický protokol, implementovaný pomocí této knihovny, by měl příliš dlouhou odezvu, což není žádoucí. Schopnost knihovny spočíst v tomto testu $a = 2^{256} - 1$ je potvrzení teze, že knihovna neumí ukládat ve svém datovém typu čísla maximálně o 256 bitech, avšak v rámci práce s nimi zvládá i větší hodnoty, byť ne stabilně. Dle zkoumání vnitřních funkcí je to způsobeno přístupem ke zpracovávání čísel. Uvnitř funkcí se pracuje s ukazateli, ne přímo s proměnnými.

V souhrnu se ukázalo, že zařízení Arduino UNO a Arduino NANO jsou výkonnostně stejná a lze tvrdit, že výsledky měření modelu NANO jsou stejné jako u modelu UNO a opačně. Jaké bitové délky čísel podporuje knihovna na architektuře AVR je možné najít v tabulce 3.4.

Měření na architektuře ARM

Po kompilaci a spuštění na zařízení Arduino Due bylo zjištěno, že omezení knihovny jsou stanoveny i použitou architekturou. Zatímco na zařízeních s architekturou AVR nebylo možné pracovat s 512 bitovým číslem, architektura ARM to umožňovala. Pravděpodobně by bylo možné pracovat i s číslem $a = 2^{1024} - 1$, jakožto největším 1024 bitovým číslem.

Tab. 3.4: Srovnání podporovaných délek celých čísel pro architekturu AVR (knihovna *BigNumber*)

Velikost čísla	Modulární redukce	Modulární násobení	Modulární mocnění
8 bitů	ANO	ANO	ANO
16 bitů	ANO	ANO	ANO
32 bitů	ANO	ANO	ANO
64 bitů	ANO	ANO	ANO
128 bitů	ANO	ANO	ANO
256 bitů	ANO	ANO	NE
512 bitů	ANO	NE	NE
1024 bitů	NE	NE	NE
2048 bitů	NE	NE	NE

Některé výpočty byly dokonce tak rychlé, že nebylo možné čas změřit a bylo tedy zaznamenáno 0 ms. Výsledky je možné vidět v tabulce 3.5 a podporovaným délkám čísel se věnuje přehledová tabulka 3.6.

Tab. 3.5: Výsledky modulárních operací pro architekturu ARM (knihovna *BigNumber*)

Operace	16 bitů	32 bitů	64 bitů	128 bitů	256 bitů	512 bitů
Mod. redukce	0 ms	0 ms	0 ms	0 ms	0 ms	1 ms
Mod. násobení	0 ms	0 ms	0 ms	3 ms	8 ms	26 ms
Mod. mocnení	5 ms	17 ms	78 ms	492 ms	3646 ms	25712 ms

Shrnutí a porovnání napříč architekturami AVR a ARM nabízí grafy 3.4.

3.4.2 Práce s eliptickými křivkami

Pro práci s eliptickými křivkami se dle průzkumu repositářů hodí nejvíce knihovna *micro-ecc*, která obsahuje podporu pro operace na eliptických křivkách a přímo implementuje algoritmy ECDH i ECDSA. ve výchozím stavu neumožňuje přistupovat k vnitřním funkcím. Knihovna nepodporuje operace bilineárního párování ani křivky přátelské k párování.

Po zkoumání zdrojového kódu byl zjištěn postup pro *„odemčení“* funkce umožňující násobení bodu na křivce skalárem. Knihovna má předpřipravené tyto eliptické křivky: *secp160r1*, *secp192r1*, *secp224r1*, *secp256r1* a *secp256k1*.

Tab. 3.6: Srovnání podporovaných délek celých čísel pro architekturu ARM (knihovna *BigNumber*)

Velikost čísla	Modulární redukce	Modulární násobení	Modulární mocnění
8 bitů	ANO	ANO	ANO
16 bitů	ANO	ANO	ANO
32 bitů	ANO	ANO	ANO
64 bitů	ANO	ANO	ANO
128 bitů	ANO	ANO	ANO
256 bitů	ANO	ANO	ANO
512 bitů	ANO	ANO	ANO
1024 bitů	ANO	ANO	ANO
2048 bitů	ANO	ANO	NE

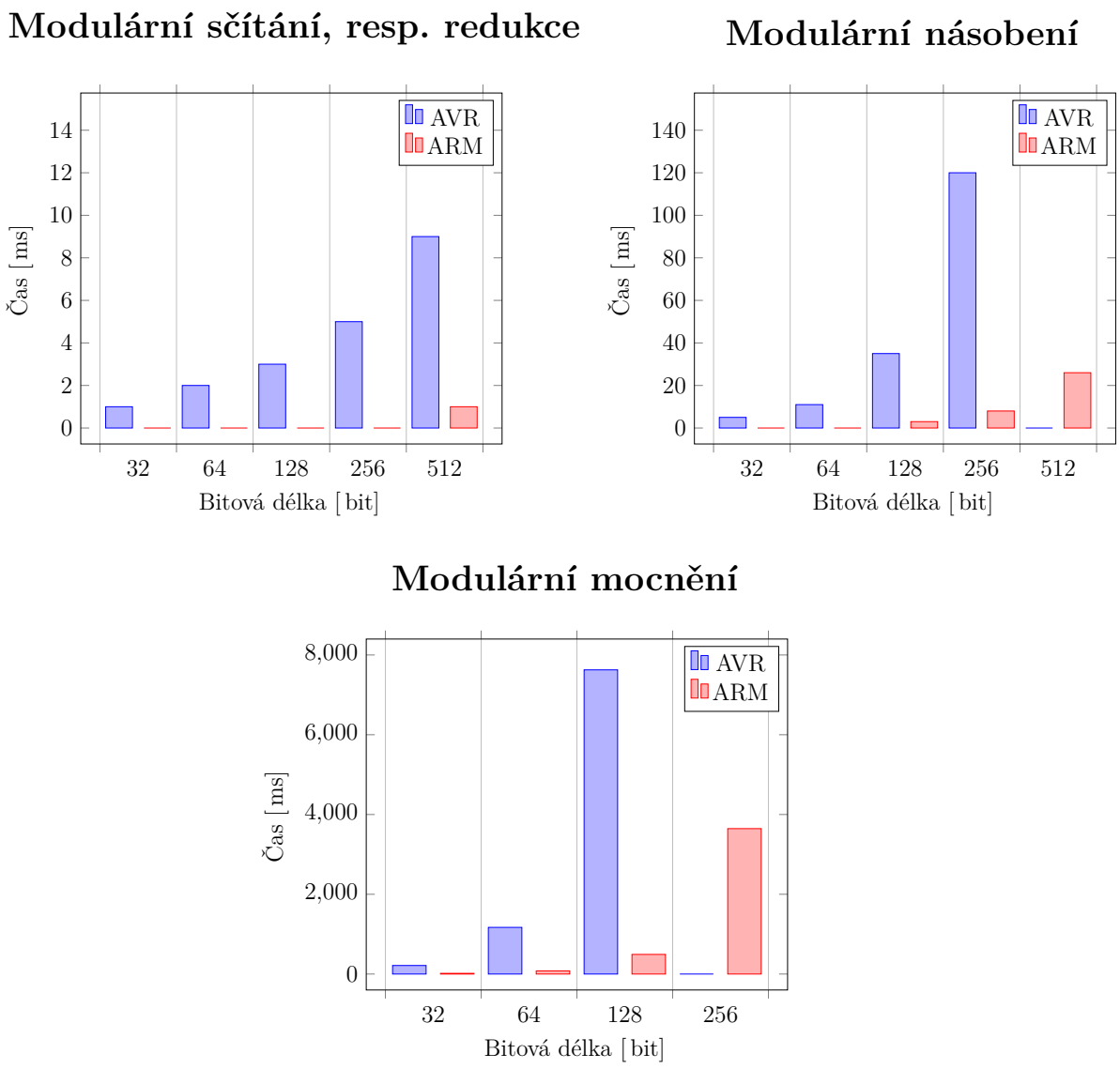
K měření výkonu práce s eliptickými křivkami se hodí algoritmy ECDH a ECDSA integrované v *micro-ecc*. Při využití dostupných eliptických křivek lze dojít k důležitým výsledkům. Co se ECDH týče, vytvoří se dvě entity, které mezi sebou chtějí vytvořit zabezpečený kanál a tedy mít jeden společný klíč. Pokud je klíč na konci procesu stejný pro obě entity, byl test úspěšný. Pro testování ECDSA byl zvolen postup o dvou krocích: podepsání a verifikace podpisu. Je-li verifikace podpisu úspěšná, algoritmus fungoval v pořádku a je možné prohlásit test za úspěšný.

Úprava knihovny

Pro měření rychlosti operací s body na eliptických křivkách, musela být knihovna dodatečně upravena, protože neobsahuje žádný přístupový bod k operaci sčítání bodů, ovšem k násobení již ano. Dle algoritmu ECDSA dochází ke sčítání bodů v procesu verifikace digitálního podpisu. Byla tedy upravena funkce kontroly podpisu tím způsobem, že byla pokrácena o části nedůležité pro operaci sčítání bodů, což je možné vidět na výpisu 3.2. Proces násobení bodů se nachází v ECDH. Následně byla provedena měření.

Výpis 3.2: Část kódu ECDSA věnující se sčítání bodů

```
uECC_vli_set(sum, _public, num_words);
uECC_vli_set(sum + num_words, _public + num_words, num_words);
uECC_vli_set(tx, curve->G, num_words);
uECC_vli_set(ty, curve->G + num_words, num_words);
uECC_vli_modSub(z, sum, tx, curve->p, num_words);
XYcZ_add(tx, ty, sum, sum + num_words, curve);
uECC_vli_modInv(z, z, curve->p, num_words);
apply_z(sum, sum + num_words, z, curve);
```



Obr. 3.4: Grafy modulárních operací (knihovna *BigNumber*)

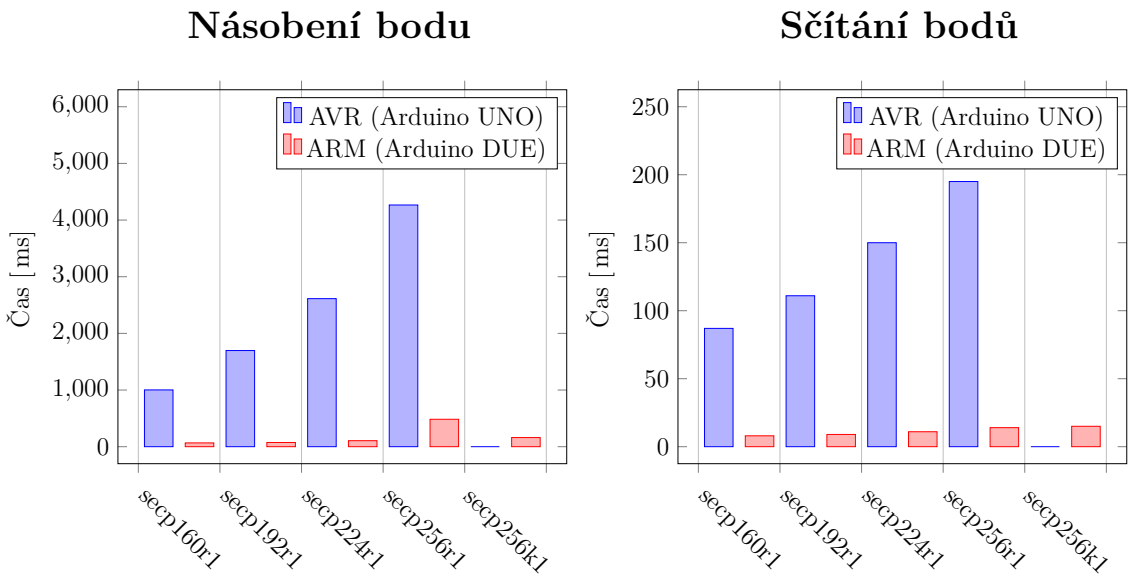
Měření

Sčítání bodů bylo relativně rychlé na obou architekturách (AVR i ARM). u AVR bylo naměřeno maximálně 195 milisekund v případě použití *secp256r1* a na architektuře ARM to bylo pouze 14 milisekund. Je tedy zřejmé, že AVR se ani na tuto operaci příliš nehodí, ovšem nedopadlo o tolik hůře v porovnání s ARM jako v jiných testech. Pakliže je použito násobení bodu na křivce, jedná se o náročnější operaci, než-li je sčítání. To vede k faktu, že u měření násobení se rozdíly mezi AVR a ARM více prohloubily. Násobení na *secp256r1* trvalo na AVR téměř 5 vteřin, což je nepřijatelný čas pro reálnou implementaci, ARM ale na téže křivce dosáhl času 484 milisekund.

Výsledky sčítání i násobení na obou platformách nabízí souhrnná tabulka 3.7. Data byla též vynesena do grafů 3.5.

Tab. 3.7: Výsledky operací s body na eliptické křivce

Architektura AVR					
Operace	secp160r1	secp192r1	secp224r1	secp256r1	secp256k1
Násobení	1002 ms	1697 ms	2613 ms	4266 ms	nelze
Sčítání	87 ms	111 ms	150 ms	195 ms	nelze
Architektura ARM					
Operace	secp160r1	secp192r1	secp224r1	secp256r1	secp256k1
Násobení	67 ms	74 ms	106 ms	484 ms	161 ms
Sčítání	8 ms	9 ms	11 ms	14 ms	15 ms



Obr. 3.5: Grafy operací s body na eliptické křivce (knihovna *micro-ecc*)

3.4.3 ECDH a ECDSA

Měření na architektuře AVR

V rámci testování ECDH a ECDSA algoritmů obsažených v knihovně *micro-ecc* bylo na architektuře AVR zjištěno, že modely UNO a NANO nejsou schopné pracovat s křivkami *secp256k1* pro ECDH a *secp256r1* pro ECDSA. Výsledky modelů UNO a NANO jsou zaměnitelné.

Měření ECDH potvrdilo, že AVR nedisponuje dostatečným množstvím výpočetních prostředků. na křivce *secp160r1* zabral celý *„proces domluvy na klíči“* průměrně téměř 5 s, což je velice dlouhá doba. Pro křivku *secp192r1* to bylo 7 s, pro *secp224r1* 11 s a pro *secp256r1* dokonce 18 s. Tyto výsledky nepřinášejí příliš optimistické vyhlídky pro využití architektury AVR pro tyto účely (viz tabulka 3.8).

Tab. 3.8: Výsledky ECDH a ECDSA pro arichtekturu AVR

Algoritmus	secp160r1	secp192r1	secp224r1	secp256r1
ECDH	5 s	7 s	11 s	18 s
ECDSA	4 s	6 s	9 s	nelze

Měření na architektuře ARM

Při měření ECDH na křivce *secp256r1*, která by měla být nejnáročnější, co se týče výpočetních nároků algoritmu, byly výsledky okolo 800 ms – to je více než pětkrát kratší doba, než na křivce *secp160r1* na architektuře AVR. Jedná se o výsledky, které by bylo možné považovat za použitelné pro implementaci do kryptografického systému.

ECDSA přineslo překvapení v podobě možnosti využití křivky *secp256r1*. To nebylo na architektuře AVR možné – program nefungoval. Zde byl výsledek obdobný, jako u ECDH. ARM prokázal svůj výkon a pro využití ECDSA jednoznačně použitelný.

Dalším překvapivým výsledkem ECDSA na architektuře ARM jsou výsledky na křivce *secp160r1* – doby pro výpočet byly totiž delší, než pro křivku *secp256r1*. Tato chyba se projevila pokaždé, taktéž při změně pořadí testovaných křivek, přičemž implementační přístup byl obecný a stejný pro všechny křivky. Dále bylo zjištěno, že není možné využít křivku *secp256k1* pro ECDH. Zřejmě není problém jen ve výkonu – tato křivka nefungovala ani na AVR.

Jelikož se jednalo o poměrně náročný test, byly zjevné i další vlivy, které mohou ovlivnit rychlost či spolehlivost zařízení. Pro architekturu ARM to byla kvalita připojení a napájení. Během prvních testů se stalo, že použití nekvalitního kabelu

způsobilo problémy výkonnostního rázu. Nepodařilo se ovšem zjistit, zda-li to bylo interferencí s jiným kabelem, nebo nestabilitou napětí, nicméně se jednalo o opakovaný jev. I toto je ale poznatek, který je potřeba brát při návrhu připojení zařízení Arduino DUE v potaz. Měření jsou zaznamenána v tabulce 3.9.

Tab. 3.9: Výsledky ECDH a ECDSA pro arichtekturu ARM

Algoritmus	secp160r1	secp192r1	secp224r1	secp256r1
ECDH	400 ms	410 ms	560 ms	880 ms
ECDSA	800 ms	490 ms	640 ms	762 ms

3.4.4 Symetrická kryptografie a hash funkce

Samotnou implementaci kryptografických algoritmů nabízí například již zmíněná knihovna *arduinolibs* [19]. v rámci knihovny je možné najít podporu pro symetrické a asymetrické šifrovací algoritmy i další (třeba hašovací algoritmy).

Vývojáři starající se o knihovnu *arduinolibs* [22] testovací soubory pro měření a ověření funkčnosti obsažených algoritmů. Testy byly provedeny na modelech UNO (architektura AVR) a DUE (architektura ARM).

Mezi testované a měřené algoritmy v rámci této práce patří:

- **hašovací algoritmy** – SHA-512, SHA3-512, Blake2s a Blake2b,
- **blokové šifrovací algoritmy** – AES128, AES192 a AES256,
- **proudové šifrovací algoritmy** – ChaCha.

Knihovna se šetrně chová k paměti zařízení. na architektuře AVR zabíral každý test přibližně 30% programové paměti, ovšem 85% dynamické paměti.

Hašovací algoritmy SHA

Hašovací algoritmy SHA-512 a SHA-3[23] patří do stejné rodiny algoritmů – SHA (*Secure Hash Algorithm*). Tyto algoritmy by z povahy věci měly být rychlé a především splňovat podmínky (pro bezpečné hašovací algoritmy[24]: jednosměrnosti, bezkoliznosti, stejné délky výstupu pro jakýkoliv vstup, nemožnosti hledat vstupy systematicky a faktu, že není žádná korelace mezi vstupními a výstupními daty. Totéž platí i pro algoritmy Blake2s a Blake2b.

V tomto měření byl porovnán starší algoritmus SHA-512 s novějším algoritmem SHA3-512 (označovaný též *Keccak*).

Zatímco hašovací algoritmus SHA-2 byl vyvíjen jako *closed-source*, SHA-3 je *open-source*. Samotní autoři algoritmu přirovnávají SHA-3 k šifrovacímu algoritmu

AES v tom slova smyslu, že se jedná o *open-source* alternativu k SHA-2 stejně tak, jako AES je *open-source* alternativa k algoritmu 3DES [25].

Během tohoto měření se znovu ukázal výkonnostní rozdíl mezi ARM a AVR. Při samotném hašování byl ARM více než čtyřicetkrát rychlejší a ve fázi finalizace je na tom dokonce ještě lépe, viz tabulka 3.10.

Tab. 3.10: Výkonnostní testy algoritmů SHA-512 a SHA3-512

SHA-512		
Architektura	Hašování (kB/s)	Finalizace (op./s)
AVR	7,6	59
ARM	341	2650
SHA3-512		
Architektura	Hašování (kB/s)	Finalizace (op./s)
AVR	9	123
ARM	93	1302

Porovnání architektury AVR s architekturou ARM přineslo i u tohoto měření obdobné výsledky. Opět ARM překonal AVR, ale zde byl ARM o trochu více než desetkrát rychlejší. Výsledek 93 kb/s není vzhledem k charakteru zařízení tak špatný.

Přinejmenším jedno zajímavé zjištění tato měření přinesla. Zatímco architektura AVR byla rychlejší při provádění algoritmu SHA3-512 než při provádění SHA-512, v případě architektury ARM to bylo naopak, jak je vidět v tabulce 3.10.

Blake2s a Blake2b

Hašovací funkce Blake2[26] má být rychlejší alternativou pro všechny generace SHA, ovšem zároveň přinejmenším stejně bezpečná jako SHA-3. Byla popsaná v RFC 7693[27] a vydaná na GitHubu pod licencí CC0.

Existují dvě varianty:

- **Blake2s** – verze optimalizovaná pro 8 a 16 bitové procesory,
- **Blake2b** – varianta optimalizovaná pro 64 bitové procesory.

Blake2s na architektuře AVR (8 bitů), pro jež by měla být nejvýkonnější, dosáhl výsledku sic horšího než na architektuře ARM, ale o mnoho lepšího než v případě SHA-512 i SHA3-512. Situaci vyobrazuje tabulka 3.11.

Výkonnost Blake2b je o poznání horší než u Blake2s. To se týká obou architektur a je to přímý důkaz o rozdílném zaměření obou variant. Byť v případě samotného hašování se u architektury ARM jedná o pokles o čtyřicet procent, architektura AVR zaznamenala propad přibližně o osmdesát procent, viz tabulka 3.11. Autoři

Tab. 3.11: Výkonnostní testy algoritmů Blake2s a Blake2b

Blake2s		
Architektura	Hašování (kB/s)	Finalizace (op./s)
AVR	49	629
ARM	1252, 4	15062
Blake2b		
Architektura	Hašování (kB/s)	Finalizace (op./s)
AVR	13, 6	106
ARM	775	6012

tedy vytvořili algoritmus výkonnější než je SHA. Jsou zde též znatelné rozdíly mezi AVR a ARM a také se potvrdila tvrzení o optimalizaci obou existujících variant.

Blokové šifrovací algoritmy

Knihovna podporuje šifrování pomocí AES [28], který využívá algoritmus Rijndael [29]. Algoritmus Rijndael vyhrál v roce 2010 a velice rychle začal nahrazovat 3DES i DES ve všech oblastech. Jedná se bezpečný symetrický šifrovací algoritmus využívající 128, 256 nebo 512 bitový klíč [30].

Měření probíhalo v režimu ECB (*Electronic CodeBook*). Jedná se o nejzákladnější mód, ve kterém se šifrované bloky dat navzájem neovlivňují (nejsou provázány) a i z tohoto důvodu by se teoreticky mělo jednat o nejrychlejší režim[31]. Dále existují módy CBC (*Cipher Block Chaining*), CFB (*Cipher Feedback*), OFB (*Output Feedack*) a CTR (*Counter*).

Tab. 3.12: Výkonnostní testy algoritmů AES pro AVR i ARM

Proces	AVR			ARM		
	AES128	AES192	AES256	AES128	AES192	AES256
Šifrování (kB/s)	30	25	21, 4	151, 8	126	107, 5
Dešifrování (kB/s)	13, 8	11, 4	9, 7	87, 7	72, 3	61, 5

U výpočtů AES je ARM přibližně pětkrát rychlejší než AVR. Výkonnostní rozdíl mezi 128 a 256 bitovou verzí algoritmu činí zhruba třicet procent pro šifrování i dešifrování. To platí pro obě architektury (viz tabulka 3.12).

Z hlediska bezpečnosti v kombinaci s rychlostí by asi bylo nejlepší použít AES192. Nabízí vyšší bezpečnost s cenou patnácti procent rychlosti AES128. Faktem avšak zůstává, že dle společnosti NIST by měl AES128 být s jistotou bezpečný nejméně

do roku 2030, tudíž pokud by požadavkem byla čistě rychlost, využití 128 bitové verze AES by nebyl problém.

Proudové šifrovací algoritmy

Celá sekce *Proudové šifrovací algoritmy* využívá jako zdroj *RFC 7905* [32].

Jako zástupce proudových šifer pro knihovnu *arduinolibs* byla vybrána šifra ChaCha (konkrétně ChaCha20). Používá 256 bitový klíč, 96 bitový *nonce* a 20 rund. Její využití lze nalézt kupříkladu u TLS (*Transport Layer Security*). Návrh šifry byl přizpůsoben pro nejvyšší rychlost softwarové implementace a mimo jiné by měla šifra být odolná proti útokům postranními kanály.

Pro srovnání lze uvézt použití 128 bitového klíče a sledovat, jestli se nějakým způsobem změní výsledná rychlost. Měření je zapsáno v tabulce 3.13.

Tab. 3.13: Výkonnostní testy algoritmů ChaCha pro AVR i ARM

Proces	AVR		ARM	
	128 bitů	256 bitů	128 bitů	258 bitů
Šifrování (kB/s)	57,09	57,09	1145,45	1145,43
Dešifrování (kB/s)	57,07	57,07	1142,18	1142,16

Na výsledcích lze vidět, že na délce klíče příliš nezáleží. Rozdíly byly zaznamenány pouze na zařízení Arduino DUE – konkrétně dvacet bajtů, což lze zanedbat. Je tedy možné vyvodit závěr, že v případě použití algoritmu ChaCha20 téměř výkonnostně nezáleží na délce použitého klíče, a tak lze jedině doporučit využití nejdelší (resp. nejsilnější) možné varianty, tedy 256 bitů.

3.5 Implementace ECDH mezi dvěma zařízeními

Po provedení testování a měření výkonnosti, byla provedena reálná implementace ECDH mezi dvěma zařízeními Arduino – konkrétně dva modely NANO.

Princip protokolu ECDH:

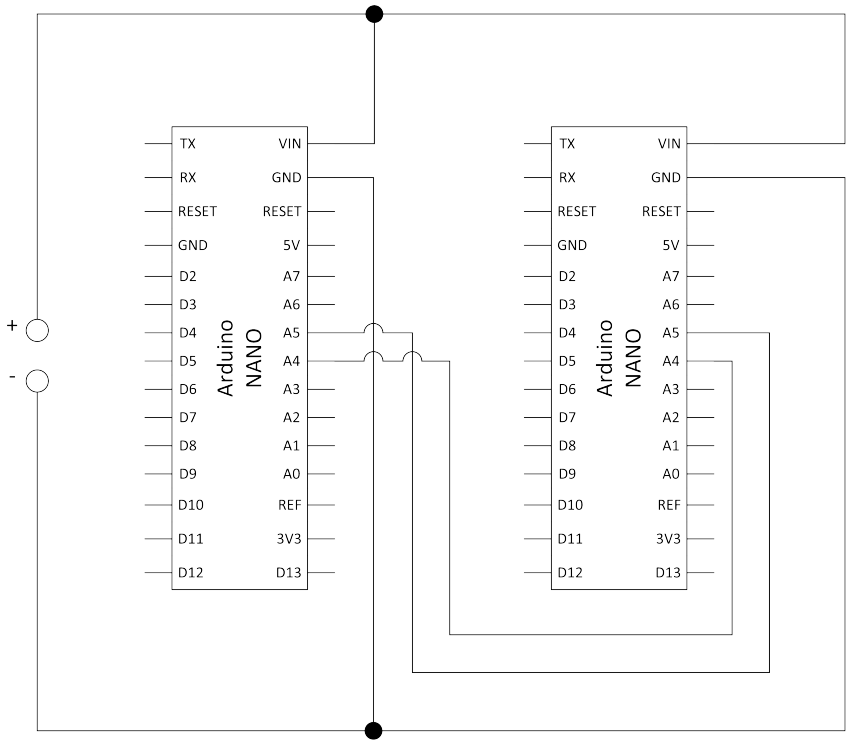
1. vytvoření veřejného klíče Q, pomocí vzorce $Q = dG$, kde $d \in \{1, n-1\}$ - jedna strana má d_A a Q_A, druhá strana má d_B a Q_B
2. výměna veřejných klíčů (Q_A a Q_B) mezi koncovými body
3. následně lze vypočíst pro jednu stranu $P = d_A Q_B$ a pro druhou stranu $P' = d_B Q_A$
4. $P = P'$, jelikož $d_A Q_B = d_b Q_A$

Pro ECDH je třeba vhodná křivka *Curve25519*, která byla poprvé představena v roce 2005 [33]. Autorem je *Daniel Julius Bernstein* – německo-americký matematik, který ve stejném roce publikoval též proudovou šifru *Salsa20* [34]. Ke standardizaci křivky došlo až v roce 2016 v rámci RFC 7748 [35]. Její popularita začala velice rychle sílit (využití našla například u síťového protokolu *Secure Shell*), díky vysoké rychlosti [36], ale i z toho důvodu, že vznikly pochybnosti ohledně bezpečnosti NIST (*National Institute of Standards and Technology*) křivek, protože jejich parametry jsou uměle vytvořené a mohly by sloužit kupříkladu NSA (*National Security Agency*) k využívání potenciálně existujících zadních vrátek. Bylo totiž odhaleno, že algoritmus *Dual_EC_DRBG* (generátor pseudonáhodných čísel) zřejmě obsahuje zadní vrátka a *Edward Snowden* obvinil NSA z masového sledování lidí prostřednictvím právě této umělé zranitelnosti [37]. Pro implementaci ECDH lze ale také použít všechny křivky obsažené v knihovně *micro-ecc*.

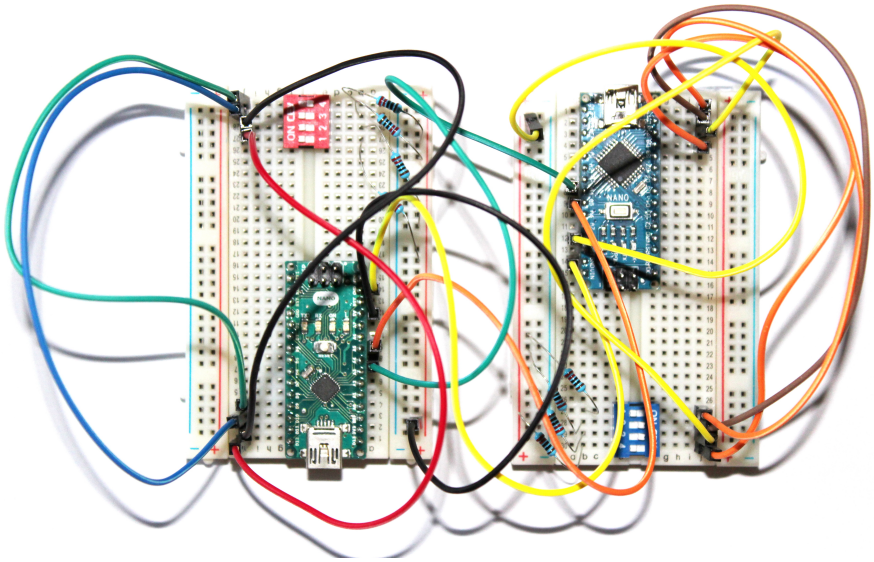
Sestava se skládala ze dvou nepájivých polí, přičemž každé bylo osazeno jedním zařízením Arduino NANO, DIP přepínačem o čtyřech možnostech nastavení a samozřejmě propojovacími kabely (popř. i rezistory). Dvěma kabely byla propojena zařízení tak, aby byl vytvořen most mezi příslušnými analogovými piny (A4↔A4 a A5↔A5). Schéma zapojení pro komunikaci je vidět na obrázku 3.6 a reálné zapojení pak na obrázku 3.7.

Komunikace mezi zařízeními byla zprostředkovaná přes sériovou sběrnici I^2C při využití modelu *MASTER-SLAVE*. Každé ze zařízeních mělo vlastní unikátní adresu na sběrnici a komunikaci inicioval vždy *MASTER*.

Po straně softwaru bylo využito pouze knihoven *micro-ecc* a *Wire* (pro zajištění podpory I^2C). s uživatelem komunikuje vytvořený program pomocí sériové linky (resp. USB s připojeným sériovým monitorem a podporou vstupu) a je totožný na obou zařízeních. Jestli bude zařízení pracovat v režimu *MASTER* či v režimu *SLAVE*, rozhoduje nastavení na prvním přepínači DIP přepínače – v poloze *„sepnuto“* se jedná o *MASTER* zařízení. Též je nutné určit, jaká křivka bude použita



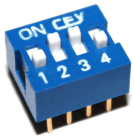
Obr. 3.6: Schéma pro komunikaci mezi zařízeními Arduino NANO



Obr. 3.7: Reálné zapojení pro implementaci ECDH mezi zařízeními Arduino NANO

a k tomu slouží zbylé tři přepínače – součet nastavených poloh *„sepnuto“* určuje použitou křivku. Přehledně jsou možná nastavení uvedena v tabulce 3.14 a konkrétní použitý DIP přepínač je na obrázku 3.8.

Po proběhnuvším nastavení začne komunikace na jejímž konci jsou shodné klíče



Obr. 3.8: Použitý DIP přepínač

Tab. 3.14: Možnosti nastavení DIP přepínače pro implementaci ECDH.

DIP přepínač			
1	2	3	4
MASTER/SLAVE	secp160r1		
MASTER/SLAVE	secp224r1		
MASTER/SLAVE	secp256r1		

dostupně na obou zařízeních. Celý proces vezme 2 až 5 s v závislosti na nastavených parametrech. Výměna informaci mezi zařízeními musí být rozložena do bloků o maximální velikost 32 bajtů – jedná se o omezení I^2C, které je nastavené knihovnou Wire.h. Vzhledem k tomu přibývá u větších přenosů také režie provozu, což přináší časové zpoždění a také větší náchylnost na chyby. Výstup na straně zařízení v režimu *MASTER*:

Výpis 3.3: Výstup ECDH v režimu MASTER

```
Registered as MASTER in MEDIUM SECURITY mode!
Please enter local address: 1
Please enter remote address: 2
Performing ECDH...
Optimizing memory...
365321915510799948223177391692441081444 62
   ↪ 06681591201006109771261036523 9
```

Výpis 3.4: Výstup ECDH v režimu SLAVE

```
Registered as SLAVE in MEDIUM SECURITY mode!
Please enter local address: 2
Performing ECDH...
Optimizing memory...
365321915510799948223177391692441081444 62
   ↪ 06681591201006109771261036523 9
```

Z výstupů jsou jasně čitelné adresy zařízení, nastavení a kroky programu. Poslední řádek každého výstupu obsahuje domluvený klíč. Jedná se o decimální zápis obou souřadnic společného bodu.

4 Praktická implementace

Tato kapitola se věnuje praktické implementaci důkazu znalosti algebraického kódu MAC, KVAC (*Keyed-Verification Anonymous Credentials*) a RKVAC (*Revocable Keyed-Verification Anonymous Credentials*) schématu. Implementace na sebe navazují a vždy rozšiřují předešlé.

4.1 Příprava knihovny

Úpravy knihovny *micro-ecc* byly nutné, aby byla zpřístupněny operace modulární aritmetiky a stejně tak i operace s body na eliptické křivce. Úpravy nijak nezasahují do již existující funkcionality, pouze ji rozšiřují. Funkce jsou obsaženy v souboru `uECC_vli.h`, resp. `uECC_vli.cpp`.

Konkrétně je tedy možné provádět modulární redukci, sčítání (resp. odčítání), násobení a inverzi. Pro účely práce s body na eliptické křivce musela být dodatečně implementována optimalizovaná funkce sčítání bodů [38], násobení bodu skalárem již knihovna obsahuje, byť je funkce před zásahem do knihovny programátorovi skryta.

Výpis 4.1: Kód pro sčítání bodů na eliptické křivce

```
//parametry p_P a p_Q představují vstupní body pro sečtení
void uECC_point_add(const uECC_word_t* p_P, const uECC_word_t* p_Q,
    ↪  uECC_word_t* p_result, const uECC_Curve curve) {
  //příprava vnitřních proměnných pro práci (eliminace zápisu do~
     ↪ vstupních proměnných)
  const wordcount_t num_words = curve->num_words;
  uECC_word_t P_x[uECC_MAX_WORDS];
  uECC_word_t P_y[uECC_MAX_WORDS];
  uECC_word_t Q_x[uECC_MAX_WORDS];
  uECC_word_t Q_y[uECC_MAX_WORDS];

  uECC_vli_set(P_x, p_P, num_words);
  uECC_vli_set(P_y, p_P+ num_words, num_words);
  uECC_vli_set(Q_x, p_Q, num_words);
  uECC_vli_set(Q_y, p_Q + num_words, num_words);

  //sčítání bodů
  XYcZ_add(P_x, P_y, Q_x, Q_y, curve);

  uECC_word_t z[uECC_MAX_WORDS];
  uECC_vli_modMult_fast(z, p_P, P_y, curve);
  uECC_vli_modInv(z, z, curve->p, num_words);
  uECC_vli_modMult_fast(z, z, P_x, curve);
```

```
22    uECC_vli_modMult_fast(z, z, p_P+num_words, curve);
23
24    apply_z(Q_x, Q_y, z, curve);
25
26    //přepis výsledku do~paměti zadané ukazatelem p_result
27    uECC_vli_set(p_result, Q_x, num_words);
28    uECC_vli_set(p_result+num_words, Q_y, num_words);
29  }
```

Při bližším zkoumání knihovny bylo odhaleno, že obsahuje části kódu, které jsou například unikátní pro každou eliptickou křivku nebo platformu. Jedná se především o optimalizující částí kódu. v případě platformy to jsou rozdílné bitové délky slov v závislosti na platformě (architektuře) a nastavení kompilátoru (lze zkompilovat a spustit 32 bitový kód na 64 bitovém zařízení, ale výsledná kompilace bude optimalizovaná právě pro 32 bitů). Pro eliptické křivky je vždy k dispozici konkrétní funkce pro modulární redukci, která je rychlejší, než univerzální funkce pro modulární redukci, kterou knihovna, jak bylo zmíněno dříve, také obsahuje. Takto upravenou knihovnu je následně možné využít pro implementaci kryptografických schémat.

Pro generování náhodných čísel lze použít nativní funkce platformy Arduino. To ovšem není vhodné především z důvodů vývoje kódu nezávislého na platformě, ale také z bezpečnosti touto cestou vygenerovaných čísel. Alternativou je použití analogového pinu, v aktuální implementaci pinu A0, pro získání náhodného čísla přes hardware, ne softwarové výpočty. na pin A0 je vhodné připojit slabý rezistor nebo volný kabel. To urychlí generování náhodných čísel. Čtením z analogového pinu lze získat data, která jsou nepředvídatelná a ovlivňuje je mnoho faktorů. Při záznamu dat je možné v průběhu času tato data porovnávat a za splněných podmínek generovat náhodná čísla.

Výpis 4.2: Kód pro generování náhodných čísel dle analogového vstupu

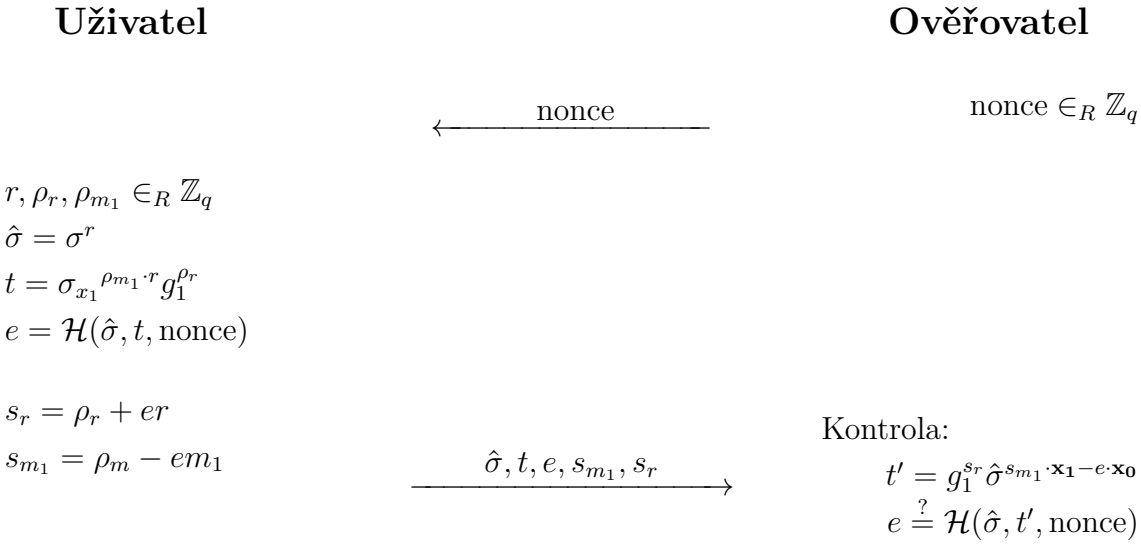
```
1  int RandomNumberGenerator(uint8_t *dest, unsigned size) {
2      while (size) {
3          uint8_t val = 0; //reprezentace hodnoty jednoho bajtu
4          for (unsigned i = 0; i < 8; ++i) {
5              int init = analogRead(0); //načtení výchozí hodnoty z~
                   ↪ pinu A0
6              int count = 0; //pomocná proměnná pro získání náhodné
                   ↪ hodnoty
7
8              //získávání náhodné hodnoty do~pomocné proměnné
9              while (analogRead(0) == init)
10                 ++count;
11
12             //finální úprava a přepis
13             if (count == 0)
```

```
                val = (val << 1) | (init & 0x01);
            else
                val = (val << 1) | (count & 0x01);
        }

        //zápis do~cílového místa v~paměti a posun na~další místo
        *dest = val;
        ++dest;
        --size;
    }

    return 1;
  }
}
```

4.2 Implementace protokolu důkazu znalosti algebraického kódu MAC

Tato kapitola byla publikována v článku na studentské konferenci EEICT 2020 [39].

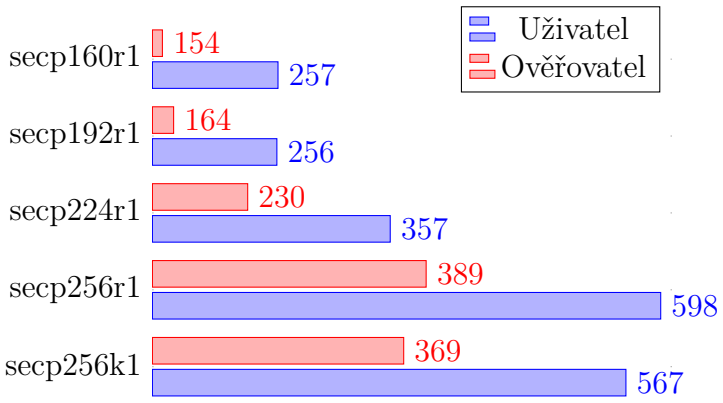
Na výkonnějším Arduino DUE byl implementován protokol s nulovou znalostí. Konkrétně se jedná o protokol dokazující znalost algebraického autentizačního kódu zprávy *MAC* (Message Authentication Code) na zprávu m (schéma je vidět na obrázku 4.1). Přičemž, jak *MAC*, tak zpráva m, zůstávají před ověřovatelem skryty.



Obr. 4.1: Důkaz znalosti autentizačního kódu $\texttt{MAC}_{wBB}$.

Tyto protokoly jsou velmi často využívány v pokročilých protokolech s ochranou

soukromí, jako jsou např. anonymní atributová pověřovací schémata nebo skupinové podpisy. Kryptografický návrh je detailně popsán v původním článku [11]. Protokol zahrnuje operace s body na eliptické křivce i modulární aritmetiku. Časová náročnost implementovaného protokolu je uvedena na obrázku 4.2.



Obr. 4.2: Časová náročnost protokolu důkazu znalosti MAC_{wBB} na zařízení Arduino DUE (časy jsou uvedené v ms)

Tento protokol nebylo možné spustit na zařízeních architektury AVR, a to ani při použití 160-bitové křivky. Pro účely implementace MAC_{wbb} byla použita upravená verze knihovny *micro-ecc*. Bez úprav mohla ovšem být použita hašovací funkce SHA256 z knihovny *arduinolibs*. Schéma je kompletně implementované v souboru `main.cpp`. Implementace není rozsáhlá, proto existence pouze jednoho souboru nečiní problém, což lze vidět na struktuře projektu v příloze A.

Výkonnostní testy pro architekturu ARM, které jsou znázorněné na grafu 4.2, ukazují, že eliptická křivka *secp160r1* dosahuje přibližně stejné náročnosti, jako křivka *secp192r1*. Jako nejvhodnější křivka se, vzhledem k rychlosti výpočtů i kryptografické *bezpečnosti, jeví secp224r1*.

Výpis 4.3: Ukázka hašování pomocí SHA256 v rámci schématu

```
SHA256 oldHash;
oldHash.reset();

uECC_word_t* resultHash = new uECC_word_t[HASH_SIZE]();
hashUpdate(sigma_A, nativeNCount * 2, byteCount * 2, &oldHash);
hashUpdate(t, nativeNCount * 2, byteCount * 2, &oldHash);
hashUpdate(nonce, nativeNCount, byteCount, &oldHash);
oldHash.finalize(resultHash, HASH_SIZE);
uECC_vli_bytesToNative(e, resultHash, byteCount);

uECC_vli_mmod(e, resultHash, n, nativeNCount);
```

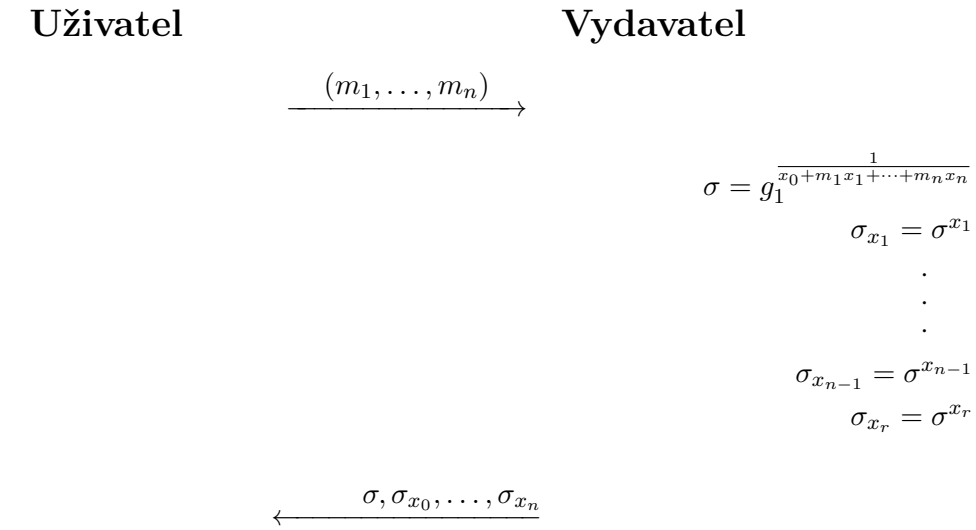
Výše uvedená sekvence kódu, jak je ve výpisu 4.3, je ekvivalentní k zápisu 4.1, který představuje hašování hodnot $\hat{\sigma}$, t a *nonce*.

$$e = \mathcal{H}(\hat{\sigma}, t, \text{nonce}) \tag{4.1}$$

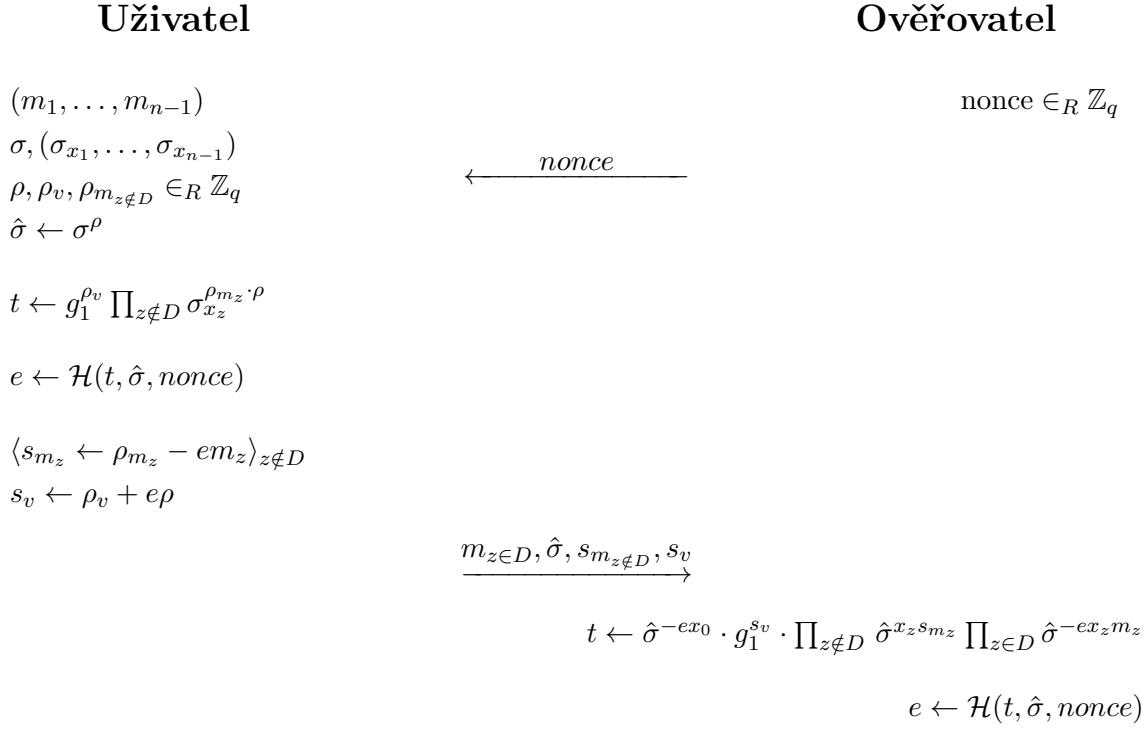
4.3 Implementace KVAC schématu

Na implementaci důkazu znalosti algebraického kód MAC_{wBB} lze postavit KVAC schéma [11]. Zatímco takovéto schéma není z výkonnostního hlediska tak problematické provozovat na počítači (či chytrém telefonu), v momentě, kdy je požadován provoz na zařízeních s omezeným výkonem, dochází ke kolizi s nedostačujícími výpočetními prostředky.

V porovnání uživatelské části s původní implementací uživatelské části KVAC schématu, jak je vidět na obrázku 4.6, je Arduino DUE rychlejší než čipová karta MultOS ML4. Tyto karty obsahují od 12 do 64KB EEPROM paměti [40] a mají hardwarovou akceleraci kryptografických operací. Hlavně jsou schopny akcelerace kryptografických algoritmů i operací na eliptických křivkách. Mají tedy oproti platformě Arduino značnou výhodu. Aplikace pro karty Multos jsou psané, stejně jako u platformy Arduino, v jazyce C, ale již je k dispozici i možnost programovat v jazyce Java. Existuje zde API, které umožňuje přistupovat k funkcím, kterými čipová karta disponuje.



Obr. 4.3: Vydávání osobních atributů KVAC schématu



Obr. 4.4: Důkaz vlastnictví osobních atributů KVAC schématu

Časovou náročnost schématu na Arduino DUE pro 10 vydaných atributů znázorňuje obrázek 4.5. Je vidět, že všechny křivky mají lineární průběh a se započtením určitých odchylek také konstantně rozdílné hodnoty od ostatních. Vydávání a práci s 10 atributy lze považovat za výpočetně náročnou činnost.

Oproti implementaci protokolu důkazu znalosti algebraického kódu MAC, je tato implementace složitější a s tím se pojí také změna struktury projektu, která je vidět na příloze B. Procesy hašování, vydávání a ověřování atributů a prokazovaní znalosti atributů jsou přítomny v souboru `wbb.hpp`, resp. `wbb.cpp`. Tento soubor zůstává stěžejní pro další implementaci uvedené v této práci. Kód implementující KVAC schéma, který se nachází v souboru `wbb.cpp`, viz obrázky 4.3 a 4.4, je rozdělen do tří hlavních částí, které jsou reprezentovány funkcemi:

1. `void issue(...)`, která se stará o proceduru vydávaní atributů,
2. `void declare(...)`, která představuje prokazovaní znalosti uživatelem,
3. `bool verify(...)`, která ověřuje důkazy znalosti získané od uživatele.

Dále byla připraven také implementace lineárního seznamu (ang. *linked list*) pro budoucí účely (především pro jednoduché a spolehlivé ukládání dat). Implementaci seznamu představují soubory `uECC_List_t.hpp`, resp. `uECC_List_t.cpp`. Aby bylo možné efektivně přenášet data o aktuální konfiguraci, tj. výběru eliptické křivky, byla doprogramována třída `uECC_Parameters_t`, jejíž konstruktor lze vidět ve výpisu 4.4.

Tato třída nejenže přenáší ukazatel na použitou eliptickou křivku, ale také umožňuje získat data o délce slov reprezentujících body na eliptické křivce v podobě čísel nebo také délce takového záznamu v bajtech.

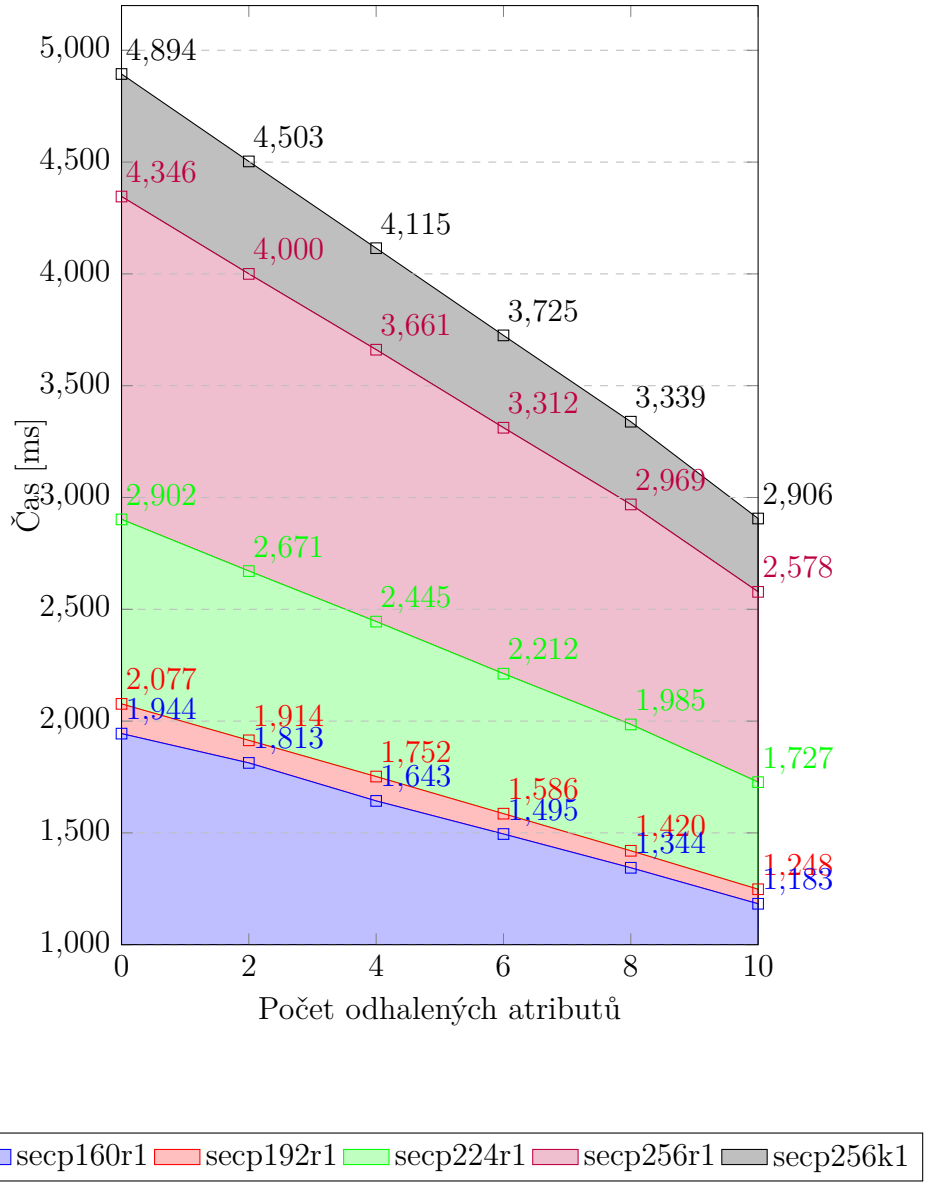
Vydávání osobních atributů KVAC schématu, jak je ve schématu 4.3, představuje podepsání atributů $(m_1, \ldots, m_n)$ vydavatelem a vytvoření podpisu atributů σ. Důkaz, že uživatel skutečně disponuje atributy, které neodkryje, se pak vytváří skrze závazek t, který je společně s hodnotami $\hat{\sigma}$ a *nonce* vložen na vstup hašovací funkce. Pro každá neodkrytý atribut se následně spočítá hodnota s_m. Ověřovatel následně zrekonstruuje hodnotu t a vloží ji, opět společně $\hat{\sigma}$ a *nonce*, na vstup hašovací funkce. Je-li výsledek stejný, jako u uživatele, uživatel je úspěšně autentizován.

Výpis 4.4: Kód třídy uECC_Parameters_t

```
1  //fromCurve udává vstupní křivku
2  uECC_Parameters_t::uECC_Parameters_t(const uECC_Curve_t *fromCurve)
3  {
4      curve = fromCurve;
5
6      //curve, n, g, nativeCount, nativeNCount a byteCount jsou po
       ↪ instanciaci veřejně dostupné hodnoty
7      n = uECC_curve_n(curve);
8      g = uECC_curve_G(curve);
9      nativeCount = uECC_curve_num_words(curve);
10     nativeNCount = uECC_curve_num_n_words(curve);
11     byteCount = uECC_curve_num_bytes(curve);
12 }
```

Výsledný kód je z velké části přenositelný na jakékoliv zařízení libovolné platformy. Výjimku tvoří funkce ovládající komunikaci zařízení Arduino s počítačem přes sériovou linku. Tento kód se nachází v souboru `main.cpp`.

Ačkoliv disponují čipové karty MultOS ML4 hardwarovou akcelerací kryptografických výpočtů, je Arduino DUE přibližně o dvacet procent rychlejší. To vyplývá z grafů na obrázku 4.6. Měření pro Arduino DUE byla prováděna desetkrát a výsledkem byl vždy aritmetický průměr všech měření.

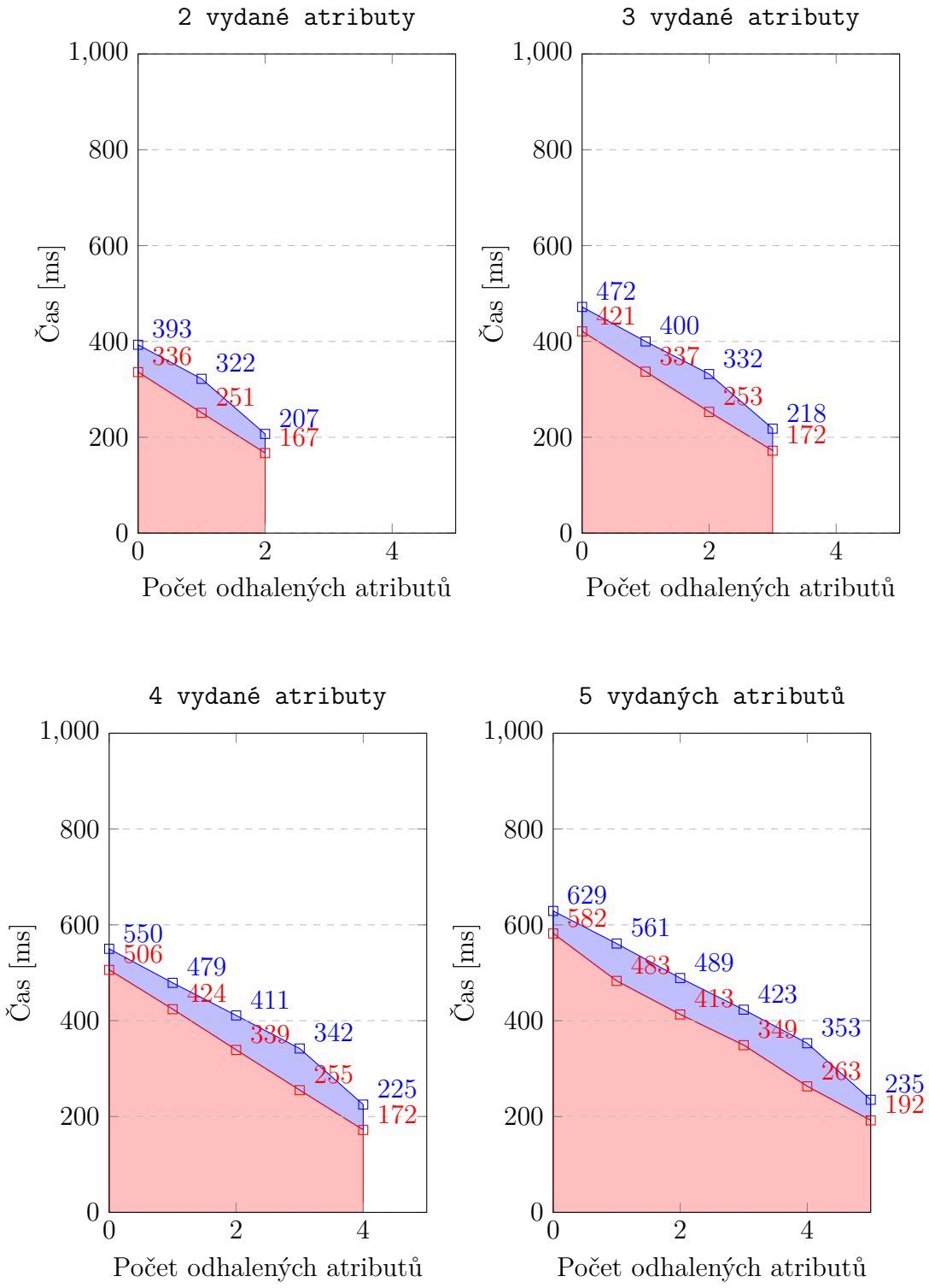


Obr. 4.5: Časová náročnost KVAC schématu na všech podporovaných křivkách.

4.4 Implementace RKVAC schématu

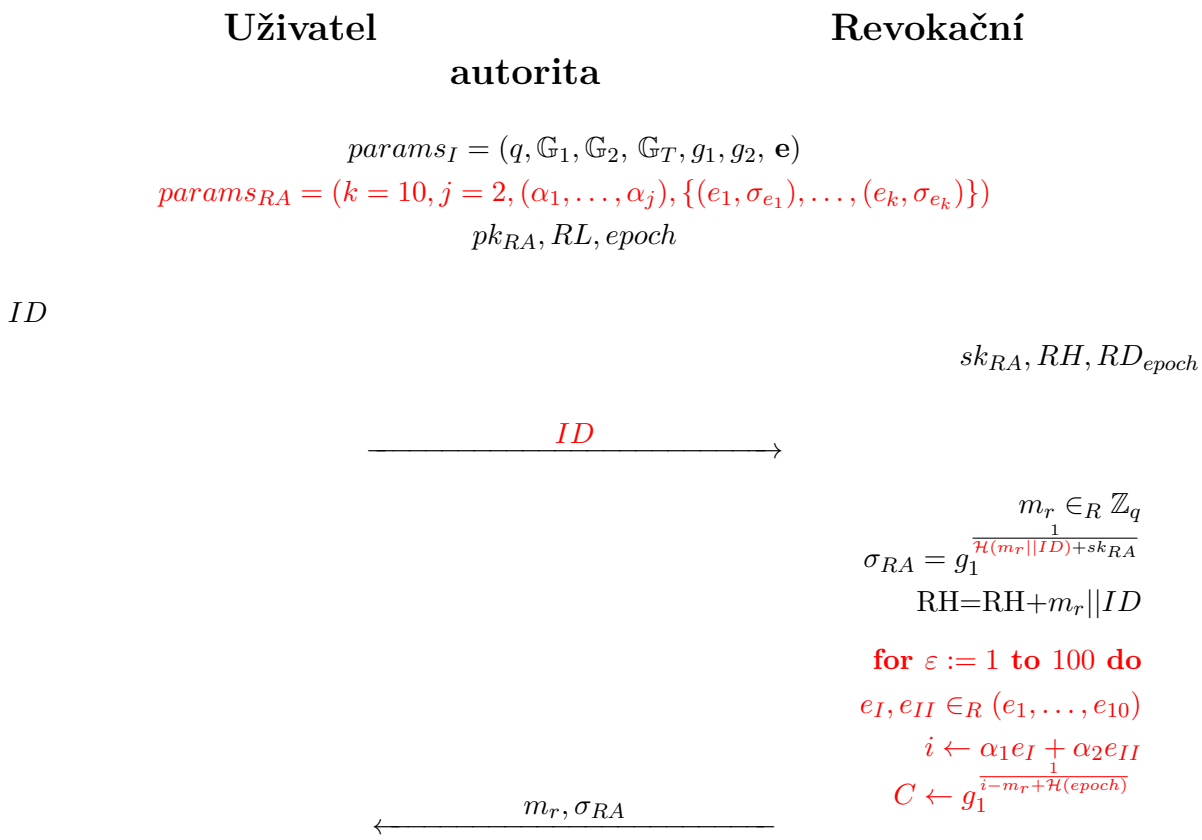
Tato kapitola se věnuje praktické implementaci protokolu RKVAC navrženého v rámci projektu Právní a technické prostředky pro ochranu soukromí v kyberprostoru, r.č. TL02000398 a řešený univerzitními týmy z VUT, MU a průmyslovým partnerem IMA v Praze. Návrh protokolu vychází z publikací řešitelského týmu [11] a [12].

RKVAC schéma je oproti KVAC schématu rozšířeno o existenci revokační autority, tudíž jako základ mohl být použit zdrojový kód KVAC schématu. Rozšíření se na kódu projevilo zvětšením jeho velikosti o téměř 20 kB a zkompilovaný program po nahrání na zařízení Arduino DUE zabíral přibližně 32 kB. Během činnosti programu lze očekávat obsazení 2,6 kB operační paměti.



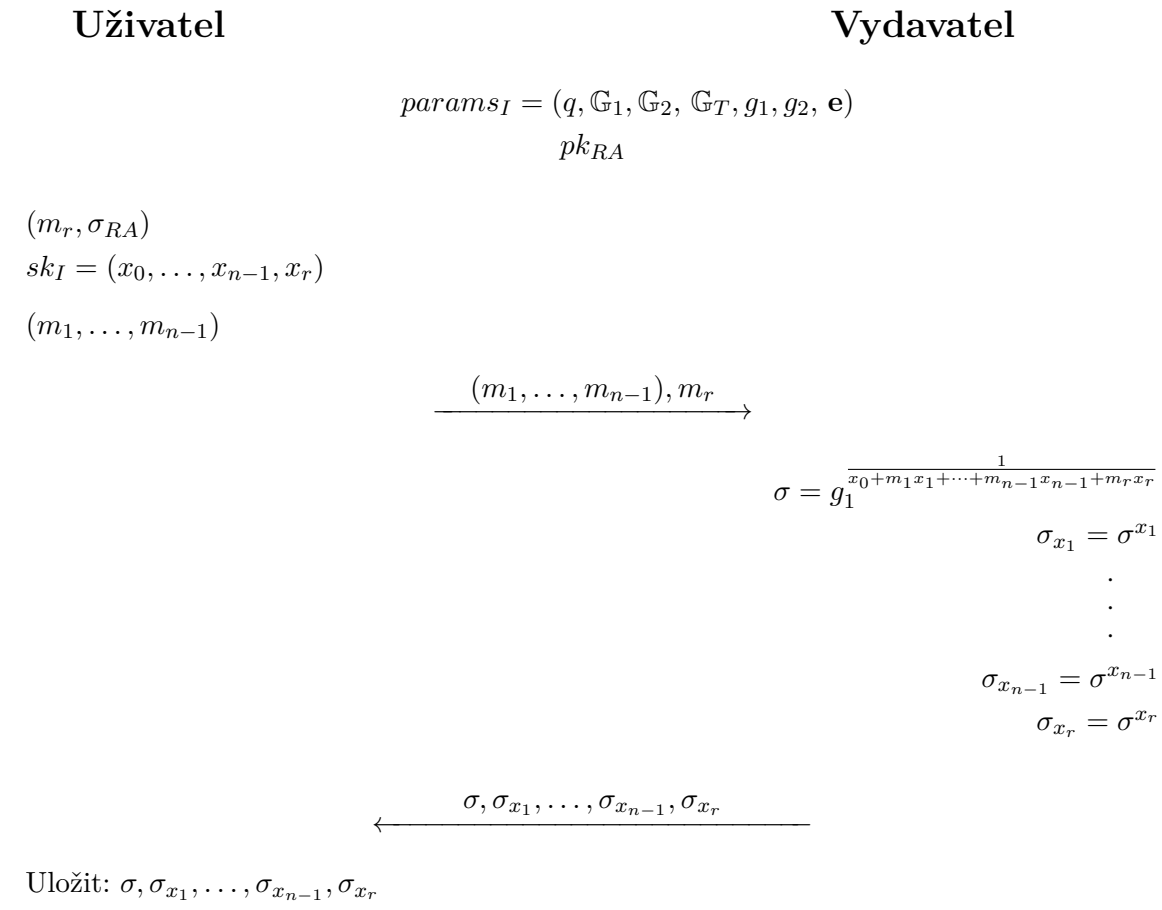
Obr. 4.6: Výkonové srovnání algoritmu KVAC na čipové kartě MultOS ML4 [1] a ARM Cortex-M3 (Arduino DUE). Modře – MultOS ML4; červeně – ARM Cortex-M3.

Kompletní implementace RKVAC by měla obsahovat též bilineární párování na eliptických křivkách. Toho ovšem nebylo možné dosáhnout, protože neexistuje knihovna pro Arduino, která by to umožňovala.



Obr. 4.7: Vydávání revokačních handlerů RKVAC schématu

Struktura kódu zůstala nezměněna a odpovídá struktuře uvedené v příloze B. Změnil se pouze obsah hlavních funkcí a samozřejmě přibilo několik funkcí pomocných, mezi které lze zařadit například generovaní revokačních handlerů. Hlavní funkce pro RKVAC obsahují kód, oproti původnímu kódu pro KVAC, kód implementující práci s revokační autoritou. Obrázek 4.10 zobrazuje časovou náročnost RKVAC schématu při vydaných 10 atributech na všech podporovaných eliptických křivkách. Kompletní schéma rozdělené do tří částí je vidět na schématech 4.7, 4.8 a 4.9. Obrázek 4.7 popisuje Vydávání revokačních handlerů umožňující revokaci uživatele ze systému. Uživatel zasílá svůj jedinečný identifikátor ID revokační autoritě, která k tomuto ID přiřadí jedinečný revokační handler m_r. Handler je revokační autoritou podepsán a spolu s podpisem $sigma_{RA}$ zaslán zpět uživateli. Současně dochází ke generování náhodných hodnot pro e_I a e_II pro uživatele. v další fázi, na obrázku 4.8, se pracuje se soukromými atributy $(m_1, \ldots, m_{n-1})$. Výsledkem je



Obr. 4.8: Vydávání osobních atributů RKVAC schématu

podpis vydavatele na atributy σ. na obrázku 4.9 je schéma dokazování disponibility atributů uživatelem. Jsou spočítány závazky t_{verify} pro skryté atributy, pak t_{revoke}, t_{sig}, t_{sigI} a t_{sigII}. Tyto hodnoty jsou spolu s dalšími vloženy na vstup hašovací funkce. Pro skryté atributy se také spočítá hodnoty s_m. Atributy odkryté se přímo předávají ověřovateli, který pak rekonstruuje závazky t. Hodnoty jsou poté opět vloženy na vstup hašovací funkce. Je-li haš na straně ověřovatele stejný, jako ten, který vypočítal uživatel, uživatel byl úspěšně autentizován.

Na obrázku 4.11 lze vidět srovnání uživatelské části s uživatelskou částí KVAC schématu. Je zřejmé, že kvůli rozšíření časová náročnost vzrostla více než trojnásobně.

Uživatel **Ověřovatel**

$$params_I = (q, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, \mathbf{e})$$
$$pk_{RA}, RL$$

$params_{RA} = (k = 10, j = 2, (\alpha_1, \ldots, \alpha_j), \{(e_1, \sigma_{e_1}), \ldots, (e_k, \sigma_{e_k})\})$

$(m_1, \ldots, m_{n-1}, m_r)$ $sk_I =$

$(x_0, \ldots, x_{n-1}, x_r)$

$\sigma, (\sigma_{x_1}, \ldots, \sigma_{x_{n-1}}, \sigma_{x_r})$

$\xleftarrow{nonce, epoch}$

$e_I, e_{II} \in_R (e_1, \ldots, e_k)$

$\sigma_{e_I}, \sigma_{e_{II}} \in_R (\sigma_{e_1}, \ldots, \sigma_{e_k})$

$i \leftarrow \alpha_1 e_I + \alpha_2 e_{II}$

$C \leftarrow g_1^{\frac{1}{i - m_r + \mathcal{H}(epoch)}}$

$\rho, \rho_v, \rho_i, \rho_{m_r}, \rho_{m_{z \notin D}}, \rho_{e_I}, \rho_{e_{II}} \in_R \mathbb{Z}_q$

$\hat{\sigma} \leftarrow \sigma^{\rho}$

$\hat{\sigma_{e_I}} \leftarrow \sigma_{e_I}^{\rho}, \hat{\sigma_{e_{II}}} \leftarrow \sigma_{e_{II}}^{\rho}$

$\bar{\sigma_{e_I}} \leftarrow \hat{\sigma_{e_I}}^{-e_I} g_1^{\rho}, \bar{\sigma_{e_{II}}} \leftarrow \hat{\sigma_{e_{II}}}^{-e_{II}} g_1^{\rho}$

$t_{verify} \leftarrow g_1^{\rho_v} \sigma_{x_r}^{\rho_{m_R} \cdot \rho} \prod_{z \notin D} \sigma_{x_z}^{\rho_{m_z} \cdot \rho}$

$t_{revoke} \leftarrow C^{\rho_{m_r}} C^{\rho_i}$

$t_{sig} \leftarrow g_1^{\rho_i} h_1^{\rho_{e_I}} h_2^{\rho_{e_{II}}}$

$t_{sigI} \leftarrow g_1^{\rho_v} \hat{\sigma}_{e_I}^{\rho_{e_I}}$

$t_{sigII} \leftarrow g_1^{\rho_v} \hat{\sigma}_{e_{II}}^{\rho_{e_{II}}}$

$e \leftarrow \mathcal{H}(t_{verify}, t_{revoke}, t_{sig}, t_{sigI}, t_{sigII}, \hat{\sigma}, \hat{\sigma_{e_I}}, \bar{\sigma_{e_I}}, \hat{\sigma_{e_{II}}}, \bar{\sigma_{e_{II}}}, C, nonce)$

$\langle s_{m_z} \leftarrow \rho_{m_z} - e m_z \rangle_{z \notin D}$

$s_v \leftarrow \rho_v + e\rho$

$s_{m_r} \leftarrow \rho_{m_r} - e m_r$

$s_i \leftarrow \rho_i + e i$

$s_{e_I} \leftarrow \rho_{e_I} - e e_I$

$s_{e_{II}} \leftarrow \rho_{e_{II}} - e e_{II}$

$\pi \leftarrow (e, s_{m_{z \notin D}}, s_v, s_{m_r}, s_i, s_{e_I}, s_{e_{II}})$

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

$t_{verify} \leftarrow \hat{\sigma}^{-e x_0} \cdot g_1^{s_v} \cdot \hat{\sigma}^{x_r s_{m_r}} \cdot \prod_{z \notin D} \hat{\sigma}^{x_z s_{m_z}} \prod_{z \in D} \hat{\sigma}^{-e x_z m_z}$

$t_{revoke} \leftarrow (g_1 C^{-\mathcal{H}(epoch)})^{-e} C^{s_{m_r}} C^{s_i}$

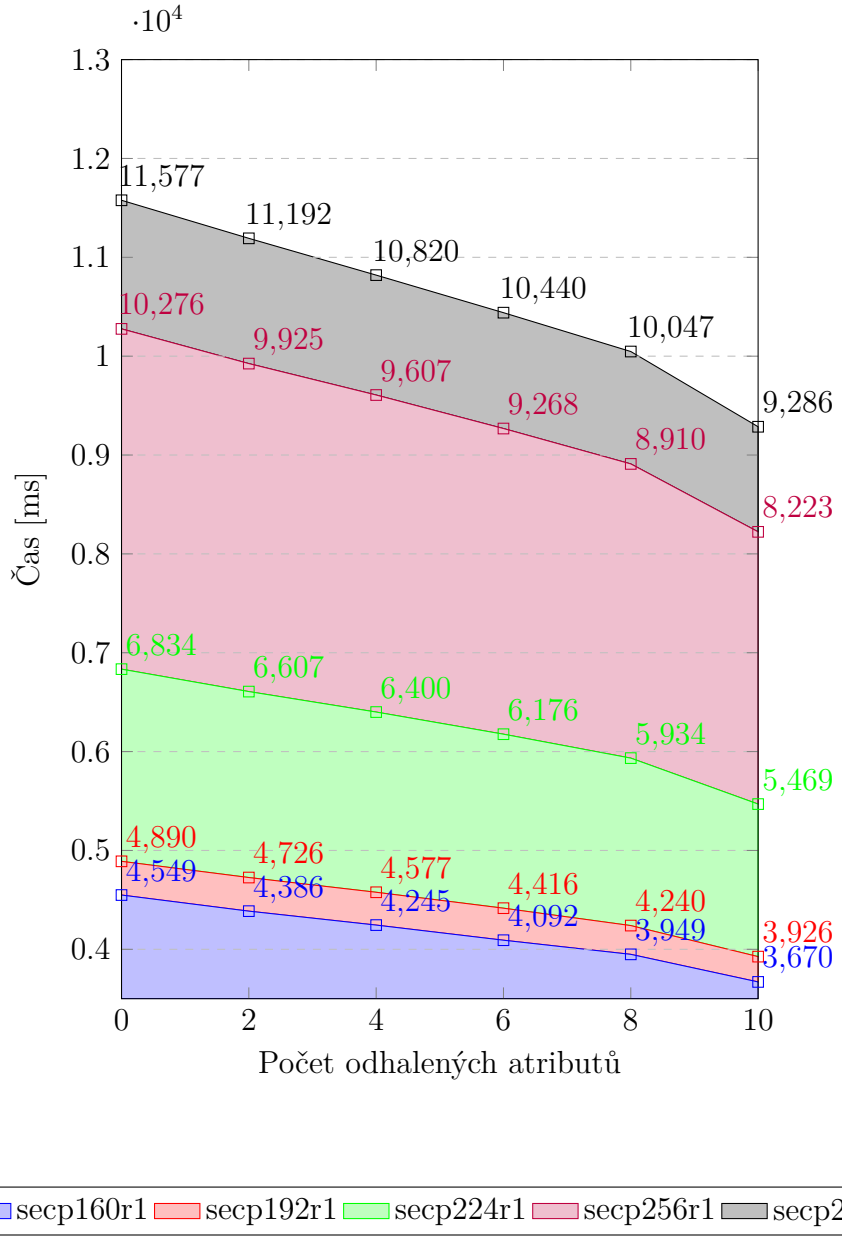
$t_{sig} \leftarrow g_1^{s_i} h_1^{s_{e_I}} h_2^{s_{e_{II}}}$

$t_{sigI} \leftarrow \bar{\sigma}_{e_I}^{-e} \hat{\sigma}_{e_I}^{s_{e_I}} g_1^{s_v}$

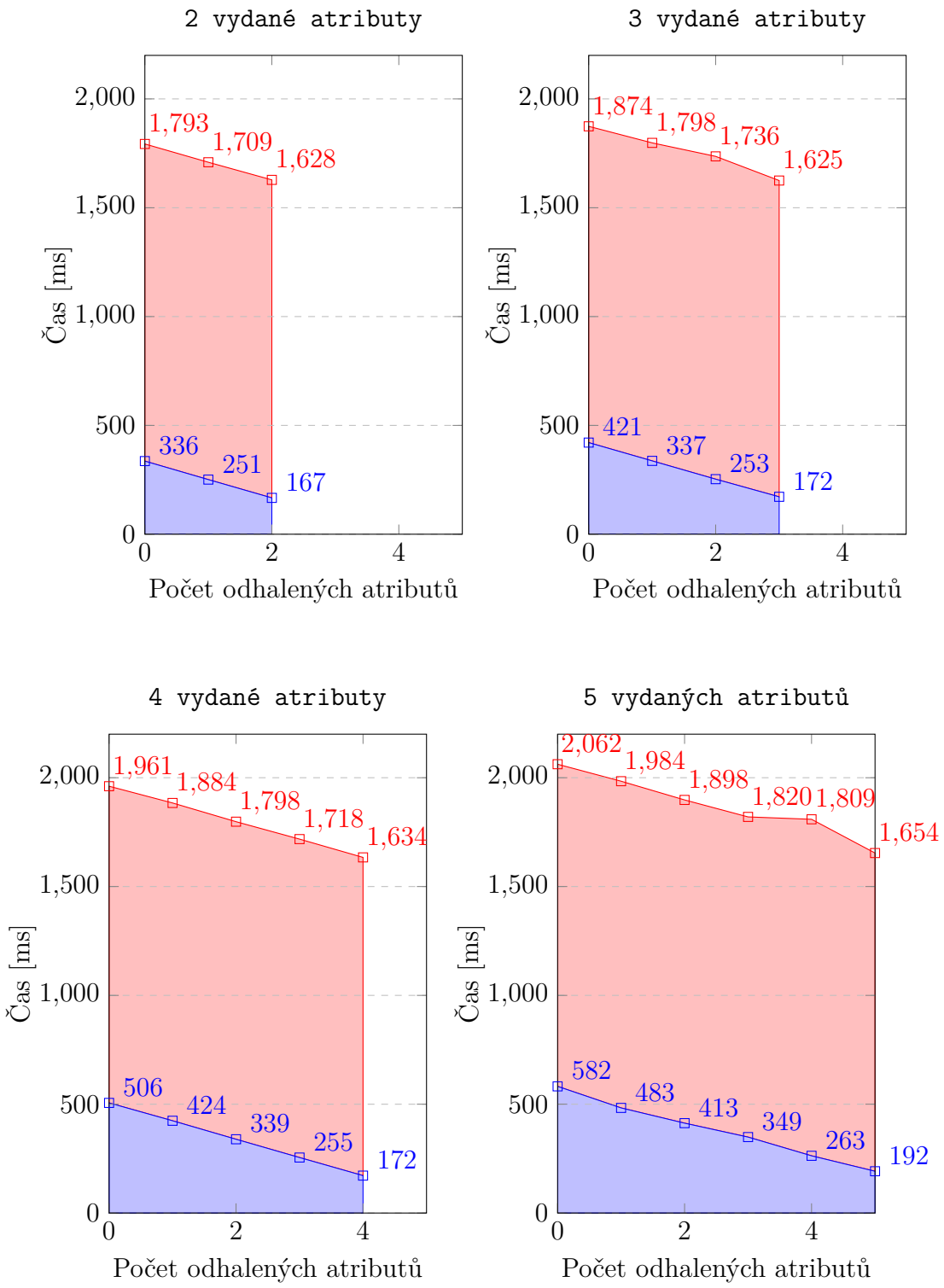
$t_{sigII} \leftarrow \bar{\sigma}_{e_{II}}^{-e} \hat{\sigma}_{e_{II}}^{s_{e_{II}}} g_1^{s_v}$

$e \leftarrow \mathcal{H}(t_{verify}, t_{revoke}, t_{sig}, t_{sigI}, t_{sigII}, \hat{\sigma}, \hat{\sigma_{e_I}}, \bar{\sigma_{e_I}}, \hat{\sigma_{e_{II}}}, \bar{\sigma_{e_{II}}}, C, nonce)$

Obr. 4.9: Důkaz vlastnictví osobních atributů RKVAC schématu



Obr. 4.10: Časová náročnost RKVAC schématu na všech podporovaných křivkách.



Obr. 4.11: Srovnání rychlosti KVAC a RKVAC schémat na Arduino DUE. Modře – KVAC schéma; červeně – RKVAC schéma.

Závěr

Tato bakalářská práce obsahuje analýzy dostupných kryptografických knihoven pro platformu Arduino pro modulární aritmetiku, operace na eliptických křivkách a implementované kryptografické funkce, jak bylo napsáno v zadání. v kapitole Praktická implementace se pak věnuje implementaci moderních atributových schémat.

Práce uvede čtenáře do problematiky ve druhé kapitole, kde jsou popsány nejpoužívanější architektury vestavěných systémů, kryptografická primitiva modulární aritmetiky i eliptických křivek. V závěru kapitoly je popsána autentizace.

Kapitola třetí se podrobně věnuje hardwaru a vývoji aplikací na platformě Arduino. Následně popisuje dostupné kryptografické knihovny, poté se věnuje jejich výkonnostním testům. Analýza dostupnosti knihoven ukázala, že nejefektivnější knihovny jsou *BigNumber* a *micro-ecc*. v rámci analýzy nebyly nalezeny žádné knihovny podporující práci s většími čísly než 512 bitů. na základě měření bylo dále zjištěno, že platforma AVR není přizpůsobena pro využití v bezpečnostních a kryptografických systémech využívajících asymetrickou kryptografii. Sekce Implementace ECDH mezi dvěma zařízeními popisuje, jakým způsobem byla provedeno experimentální zapojení dvou zařízení Arduino NANO pro účely zprovoznění ECDH s různými možnostmi nastavení eliptických křivek. Tato implementace byla postavena na knihovnách *micro-ecc* a *Wire*.

V kapitole Praktická implementace se realizuje implementaci atributových schémat na základě předtím získaných znalostí o dostupných knihovnách a výkonu platformy. Bylo využito pouze zařízení Arduino DUE, které má nejvyšší výkon a oproti zařízení postavených na AVR také schopné pracovat se všemi dostupnými eliptickými křivkami. Potřebné operace obstarala upravená knihovna *micro-ecc*, hašovací funkci SHA256 zase knihovna *arduinolibs*. Úpravy knihovny *micro-ecc* spočívali ve zpřístupnění všech dostupných vnitřních funkcích a implementaci funkce na sčítání bodů na eliptické křivce. Všechny strany protokolů, tj. uživatel, ověřovatel, vydavatel a popřípadě certifikační autorita, byly realizovány na jednom zařízení. na základě dosažených výsledků lze považovat za nejvhodnější použití eliptické křivky *secp224r1*, která je bezpečná a dosahuje akceptovatelných časů u KVAC i RKVAC schémat. Strukturu výsledných implementací lze vidět v přílohách A a B.

K práci jsou, v podobě přiloženého CD, dostupné zdrojové kódy. Médium obsahuje zdrojové kódy pro implementaci důkazu znalosti algebraického podpisu MAC, tj. soubor `uECC-MAC.zip`, pro implementaci KVAC schématu, tj. soubor `uECC-KVAC.zip`, a také pro implementaci RKVAC schématu, tj. soubor `uECC-RKVAC.zip`. Další soubor, který je na CD uložený, je soubor `README.txt`, jenž slouží pro orientaci na CD.

Literatura

[1] Ph.D. Ing. Petr Dzurenda. Kryptografická ochrana digitální identity, 2019. [online]. [cit. 13. 5. 2020]. URL: https://www.vutbr.cz/studenti/zav-prace/detail/120124?zp_id=120124.

[2] Ministerstvo průmyslu a obchodu České republiky. [online]. [cit. 20. 12. 2019]. URL: https://www.mpo.cz/assets/dokumenty/53723/64358/658713/priloha001.pdf.

[3] Arduino. URL: https://www.arduino.cc/.

[4] A brief history of arm: Part 1. [online]. [cit. 20. 11. 2019]. URL: https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/a-brief-history-of-arm-part-1.

[5] A brief history of arm: Part 2. [online]. [cit. 20. 11. 2019]. URL: https://community.arm.com/developer/ip-products/processors/b/processors-ip-blog/posts/a-brief-history-of-arm-part-2.

[6] Silicon valley historical association, 2008. [online]. [cit. 20. 12. 2019]. URL: https://www.siliconvalleyhistorical.org/atmel-history.

[7] A. J. Menezes, Van Oorschot Paul C., and Scott A. Vanstone. *Handbook of applied cryptography*. CRC Press, 2001.

[8] Jan Přikryl. Modulární aritmetika, malá fermatova věta., 2014. [online]. [cit. 20. 12. 2019]. URL: https://docplayer.cz/12377310-Modularni-aritmetika-mala-fermatova-veta.html.

[9] Eliptické křivky. [online]. [cit. 20. 12. 2019]. URL: http://brkos.math.muni.cz/files/download/elipticke-krivky.pdf.

[10] Lukáš Geyer. Eliptické křivky v kryptografii, 2018. [online]. [cit. 20. 12. 2019]. URL: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=31002.

[11] Jan Camenisch, Manu Drijvers, Petr Dzurenda, and Jan Hajny. Fast keyed-verification anonymous credentials on standard smart cards. In Gurpreet Dhillon, Fredrik Karlsson, Karin Hedström, and André Zúquete, editors, *ICT Systems Security and Privacy Protection*, pages 286–298, Cham, 2019. Springer International Publishing.

[12] Jan Camenisch, Manu Drijvers, and Jan Hajny. Scalable revocation scheme for anonymous credentials based on n-times unlinkable proofs. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society*, WPES '16, page 123–133, New York, NY, USA, 2016. Association for Computing Machinery. doi:10.1145/2994620.2994625.

[13] So you want to make an arduino. [online]. [cit. 1. 11. 2019]. URL: https://www.arduino.cc/en/main/policy.

[14] Phillip Torrone. Soapbox: My top 10 favorite arduino-compatible "clones" and derivatives, 2012. [online]. [cit. 1. 11. 2019]. URL: https://makezine.com/2012/04/24/soapbox-my-top-10-favorite-arduino-compatible-clones-and-drivatives/.

[15] Compare board specs. [online]. [cit. 1. 11. 2019]. URL: https://www.arduino.cc/en/products/compare.

[16] Getting started with the arduino due. [online]. [cit. 20. 11. 2019]. URL: https://www.arduino.cc/en/Guide/ArduinoDue.

[17] Arduino - data types. [online]. [cit. 15. 11. 2019]. URL: https://www.tutorialspoint.com/arduino/arduino_data_types.htm.

[18] Bignumber. URL: https://github.com/nickgammon/BigNumber.

[19] Arduino cryptography library. URL: https://github.com/rweather/arduinolibs.

[20] micro-ecc. URL: https://github.com/kmackay/micro-ecc.

[21] Ken MacKay. micro-ecc. [online]. [cit. 29. 11. 2019]. URL: http://kmackay.ca/micro-ecc/.

[22] Arduino cryptography library. [online]. [cit. 25. 11. 2019]. URL: http://rweather.github.io/arduinolibs/crypto.html.

[23] Sha-3. [online]. [cit. 1. 12. 2019]. URL: https://en.wikipedia.org/wiki/SHA-3.

[24] Hashovací funkce. [online]. [cit. 1. 12. 2019]. URL: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=7029.

[25] [online]. [cit. 1. 12. 2019]. URL: https://keccak.team/index.html.

[26] Blake2 — fast secure hashing, 2017. [online]. [cit. 1. 12. 2019]. URL: https://blake2.net/.

[27] Jean-Philippe Aumasson Markku-Juhani O. Saarinen. The blake2 cryptographic hash and message authentication code (mac), 2015. [online]. [cit. 1. 12. 2019]. URL: https://tools.ietf.org/html/rfc7693.

[28] Douglas Crawford. How does aes encryption work?, 2019. [online]. [cit. 1. 12. 2019]. URL: https://proprivacy.com/guides/aes-encryption.

[29] Borys Pawliw Margaret Rouse. Rijndael, 2017. [online]. [cit. 1. 12. 2019]. URL: https://searchsecurity.techtarget.com/definition/Rijndael.

[30] Keylength - nist report on cryptographic key length and cryptoperiod (2016), 2016. [online]. [cit. 1. 12. 2019]. URL: https://www.keylength.com/en/4/.

[31] Dobre Blazhevski, Adrijan Bozhinovski, Biljana Stojcevska, and Veno Pachovski. Modes of operation of the aes algorithm. 04 2013.

[32] Nikos Mavrogiannopoulos Joachim Strombergson Simon Josefsson Adam Langley, Wan-Teh Chang. Chacha20-poly1305 cipher suites for transport layer security (tls), 2016. [online]. [cit. 3. 12. 2019]. URL: https://tools.ietf.org/html/rfc7905.

[33] Daniel J. Bernstein. A state-of-the-art diffie-hellman function. [online]. [cit. 24. 11. 2019]. URL: https://cr.yp.to/ecdh.html.

[34] Daniel j. bernstein. [online]. [cit. 24. 11. 2019]. URL: https://en.wikipedia.org/wiki/Daniel_J._Bernstein.

[35] Sean Turner Adam Langley, Mike Hamburg. Elliptic curves for security. [online]. [cit. 24. 11. 2019]. URL: https://tools.ietf.org/html/rfc7748.

[36] Radek Fujdiak, Pavel Masek, Jiri Hosek, Petr Mlynek, and Jiri Misurec. Efficiency evaluation of different types of cryptography curves on low-power devices. 10 2015. doi:10.1109/ICUMT.2015.7382441.

[37] Safecurves: choosing safe curves for elliptic-curve cryptography. [online]. [cit. 24. 11. 2019]. URL: http://safecurves.cr.yp.to/rigid.html.

[38] Eccpoint_mult and point addition. [online]. [cit. 02. 06. 2020]. URL: https://github.com/kmackay/micro-ecc/issues/31.

[39] Tadeáš Cvrček. Elliptic-curve cryptography for constrained devices in internet of things and industry 4.0. In Ph.D doc. Ing. Vítězslav Novák, editor, *Proceedings of the 26th Conference STUDENT EEICT 2019*. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2020.

[40] Techical specifications. [online]. [cit. 10. 12. 2019]. URL: `http://www.multosinternational.com/en/products/multos-and-multos-stepone/technical-specifications/`.

Seznam symbolů, veličin a zkratek

ARM	Advanced RISC Machines
NVM	Non-Volatile Memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
DES	Data Encryption Standard
AES	Advanced Encryption Standard
DES	název algoritmu podle iniciál autorů Rivest, Shamir, Adleman
IDE	Integrated Development Environment
ECDH	Elliptic-Curve Diffie Hellman
ECDSA	Elliptic-Curve Digital Signature Algorithm
MAC	Message Authentication Code
KVAC	Keyed-Verification Anonymous Credentials
RKVAC	Revocable Keyed-Verification Anonymous Credentials

Seznam příloh

A Struktura kódu důkazu znalosti algebraického kódu MAC

```
uECC
├── include ..... adresář projektu specifických knihoven
├── lib ..... adresář knihoven
│   ├── Crypto ..... knihovna arduino-libs
│   └── micro-ecc ..... knihovna micro-ecc
├── src ..... zdrojové kódy
│   └── main.cpp ..... hlavní soubor
└── test ..... adresář testovacích souborů
```

B Struktura kódu KVAC a RKVAC schémat

- uECC
  - `include` — adresář projektu specifických knihoven
  - `lib` — adresář knihoven
    - `Crypto` — knihovna arduino-libs
    - `micro-ecc` — knihovna micro-ecc
  - `src` — zdrojové kódy
    - `main.cpp` — hlavní soubor
    - `types.hpp` — globální nastavení projektu
    - `uECC_List_t.cpp` — implementace lineárního seznamu
    - `uECC_List_t.hpp`
    - `uECC_Parameters_t.cpp` — implementace přístupu k parametrům křivky
    - `uECC_Parameters_t.hpp`
    - `wbb.cpp` — implementace protokolu
    - `wbb.hpp`
  - `test` — adresář testovacích souborů