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

SOFTWAROVÁ KNIHOVNA ZÁKLADNÍCH SYMETRICKÝCH A ASYMETRICKÝCH PRIMITIV MODERNÍ KRYPTOGRAFIE NA EMBEDDED PLATFORMÁCH

SOFTWARE LIBRARY OF BASIC SYMMETRIC AND ASYMMETRIC PRIMITIVES OF MODERN CRYPTOGRAPHY ON EMBEDDED PLATFORMS

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Matěj Miška

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Tomáš Lieskovan

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Bc. Matěj Miška ***ID:*** 200456

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

NÁZEV TÉMATU:

Softwarová knihovna základních symetrických a asymetrických primitiv moderní kryptografie na embedded platformách

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je provést rešerši jednodeskových platforem vhodných k nasazení ve smart meteringu a tyto platformy porovnat. Dále má student za úkol provést rešerši kryptografických algoritmů používaných v energetice. V praktické části student vytvoří nástroj na automatické testování zvolených kryptografických algoritmů a tento nástroj otestuje na alespoň třech vybraných embedded platformách blízkých platformám běžně používaných v energetice. Platformy student porovná a provede diskuzi nad výsledky.

DOPORUČENÁ LITERATURA:

[1] BRYCHTA, Josef. Srovnání kryptografických primitiv využívajících eliptických křivek na různých hardwarových platformách: Comparison of cryptographic primitives used in elliptic curve cryptograpny on different hardware platforms. 2019. PhD Thesis. Brno University of Technology.

[2] HOFFMANN, Stefan G.; MASSINK, Robin; BUMILLER, Gerd. New security features in DLMS/COSEM—A comparison to the smart meter gateway. In: 2015 IEEE Innovative Smart Grid Technologies-Asia (ISGT ASIA). IEEE, 2015. p. 1-6.

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

Vedoucí práce: Ing. Tomáš Lieskovan

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

ABSTRAKT

Tato diplomová práce se zabývá rešerší kryptografických primitiv na embedded systémech, které jsou v navazující praktické části testovány na platformách Raspberry Pi a výsledky porovnány. Obsahem rešerše je objasnění využití kryptografie v informačních systémech, příklad protokolu využívajícího kryptografii v energetice, výběr kryptografických primitiv na základě doporučení bezpečnostních institucí, popis embedded platforem Raspberry Pi a uvedení kryptografických knihoven poskytujících nástroje na provádění kryptografických operací. Výsledkem teoretické části práce je soupis potřebných informací pro vytvoření testovacího nástroje. Navazující praktická část se zaobírá teoretickým návrhem požadovaného nástroje, určením funkcí, kterými nástroj musí disponovat a způsobem provádění testování. Následuje popis výběru programovacího jazyka a vývojového prostředí vhodného pro tuto práci společně s popisem vnitřní struktury vytvořené aplikace. Parametry testování jsou především výpočetní, paměťová a časová náročnost kryptografických primitiv na systém. Závěr práce se věnuje způsobům, jak spustit a ovládat aplikaci, prezentaci naměřených výsledků, samotným výsledkům testování vybraných embedded platforem a diskusi těchto výsledků.

KLÍČOVÁ SLOVA

Kryptografická primitiva, kryptografie, DLMS/COSEM, embedded platforma, Raspberry Pi, NIST, ENISA, NÚKIB, Blokové šifry, Proudové šifry, Hash, Digitální podpis, Distribuce klíčů, Algoritmy autentizačních technik, OpenSSL, Crypto++, Python, PyCryptography, matplotlib, ECPy, eciespy, PyQt5, GUI.

ABSTRACT

This master thesis deals with a search of cryptographic primitives for embedded systems, which are tested on Raspberry Pi platforms and the results compared in the subsequent practical part. The content of the research is an explanation of the use of cryptography in information systems, an example of a protocol using cryptography in the energy sector, the selection of cryptographic primitives based on recommendations from security institutions, a description of Raspberry Pi embedded platforms and an introduction of cryptographic libraries providing tools to perform cryptographic operations. The theoretical part of the thesis results in an overview of the information needed to create a test tool. The subsequent practical part deals with the theoretical design of the required tool, the determination of the functions that the tool must have and the way of performing the testing. This is followed by a description of the choice of the programming language and development environment suitable for this work, together with a description of the internal structure of the developed application. The testing parameters are mainly the computational, memory and time requirements of the cryptographic primitives on the system. The paper concludes with methods of running and controlling the application, possible presentation of the measured results, the actual results of testing the selected embedded platforms and a discussion of these results.

KEYWORDS

Cryptographic primitives, cryptography, DLMS/COSEM, embedded platform, Raspberry Pi, NIST, ENISA, NÚKIB, Block ciphers, Stream ciphers, Hash, Digital signature, Key distribution, Algorithms of authentication techniques OpenSSL, Crypto++, Python, PyCryptography, matplotlib, ECPy, eciespy, PyQt5, GUI.

MIŠKA, Matěj. *Softwarová knihovna základních symetrických a asymetrických primitiv moderní kryptografie na embedded platformách*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2021, 113 s. Diplomová práce. Vedoucí práce: Ing. Tomáš Lieskovan

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Matěj Miška
VUT ID autora:	200456
Typ práce:	Diplomová práce
Akademický rok:	2021/22
Téma závěrečné práce:	Softwarová knihovna základních symetrických a asymetrických primitiv moderní kryptografie na embedded platformách

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. Tomáši Lieskovanovi za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Nevyhnutelným prvkem jakékoliv komunikační technické soustavy dnešní doby jsou kryptografická primitiva. Jejich využití v dnešním světě informačních systémů je téměř důležitější, než samotný provoz daného systému. Využití kryptografie je velmi rozsáhlé, avšak její hlavní bezpečnostní přínosy se dají shrnout do pojmů autentizace, autorizace, zachování důvěrnosti a umožnění nepopiratelnosti. Možnosti implementace prostředků zajišťujících tyto vlastnosti jsou však nepřeberné.

Kritéria pro výběr správné implementace jsou pro výpočetní systémy pracující s kryptografickými nástroji velmi prosté. V první řadě se jedná o obtížnost znovuzískání hledané informace, následované časem nutným k provedení výpočetní operace a paměťovou náročností vyžadovanou prováděním kryptografických úkonů.

Problém však nastává při snaze zmenšit tento výpočetní systém pro specifická využití ve větších systémech. V takovém případě se stává výpočetní síla těchto systémů slabší, čemuž je nutné přizpůsobit také implementace již zmíněných kryptografických primitiv. Samozřejmě se v tomto případě nesmí zabezpečení celého systému snížit pod úroveň považovanou za bezpečnou po dobu jeho provozu.

Výše zmíněný problém implementace kryptografických primitiv na systémy s nižší výpočetní silou je rozebrán v této diplomové práci. Semestrální část práce rozebírá hardwarové a softwarové prostředky a jejich využívání, na jejímž základě následující diplomová práce testuje vybrané poznatky pomocí vytvořeného testovacího nástroje. Konkrétněji se jedná o aplikaci, umožňující testování a porovnávání platforem na základě naměřených výsledků, pro určení nejvhodnějšího systému na implementaci protokolů využívaných v energetice. Fungování takovýchto systémů vyžaduje kryptografická primitiva pro zajištění bezpečnosti při fungování v reálném světě. Pro zkoumání způsobů a možných implementací jsou v této práci vybrána primitiva na základě doporučení a nařízení pro udržení dostatečné úrovně bezpečnosti dnešní doby.

Navazující praktická část práce se z počátku zabývá teorií a nároky na vytvářenou aplikaci. Dle zmíněných požadavků jsou následně vybrány prostředky vhodné pro tvorbu tohoto nástroje a jeho testování.

Největší část práce tvoří detailní popis vytvořené aplikace pro testování kryptografických primitiv včetně způsobů implementace zmíněných algoritmů, měření, ukládání a prezentace naměřených výsledků. Procesům spuštění a ovládání aplikace je věnována následující kapitola společně s pokyny k jejímu zprovoznění v případě výskytu problémů.

V závěru práce je provedeno testování na předem vybraných embedded platformách, jejichž výsledky jsou slovně porovnány a je provedena diskuse. Výsledky této praktické části jsou součástí přílohy práce.

1 Kryptografie v informačních systémech

Dnešní systémy pracující na výpočetních strojích mohou obsahovat velmi citlivé informace, se kterými může nakládat pouze ověřená osoba [3]. Tuto myšlenku je také možné přenést do oblasti komunikace, kde jedinými pověřenými osobami pro manipulaci s přenášenými daty jsou autentizovaní uživatelé. Tato kapitola je věnována obecnému popisu využívání kryptografie v praxi.

1.1 Důvody využití kryptografie

V reálném světě jsou informace nejcennější komoditou, a proto je nutné chránit je před entitami snažícími se dané informace získat, modifikovat, či dokonce zničit. Těmto útokům, nebo alespoň snaze o poškození či odnětí dat, se snaží koncoví uživatelé zabránit kryptografickými prostředky zajišťující mnoho cílů bezpečnosti, přičemž těmi nejdůležitějšími jsou:

- Autentizace - umožňuje jednoznačné ověření koncového uživatele na základě pravdivosti jeho atributů.
- Důvěrnost - soubor pravidel zajišťující alespoň minimální možnou ochranu před neautentizovanou manipulací s daty.
- Integrita dat - zajištění konzistence přenosu dat.
- Nepopiratelnost - vlastnost, znemožňující autora dat popřít jejich vytvoření, manipulaci, či jejich zaslání.

K naplnění těchto cílů užívá kryptografie algoritmů založených na různých přístupech k implementaci a logice utajování dat [3]. Zde se kryptografie dělí na dvě hlavní podkategorie:

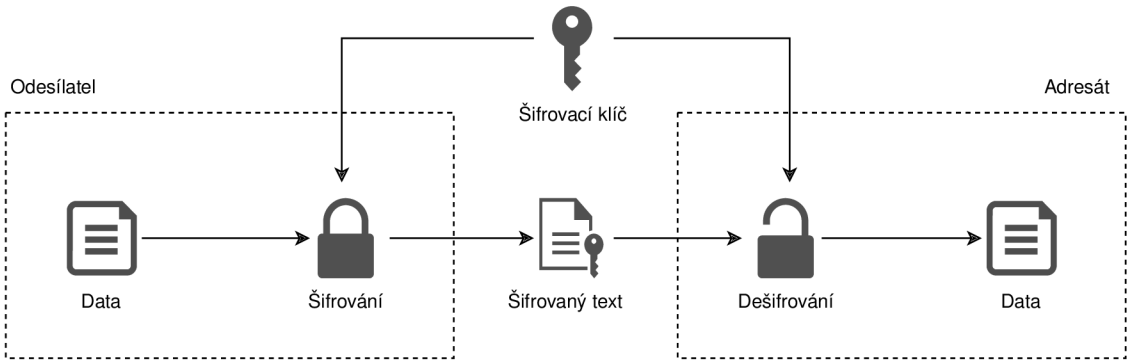
- Symetrická kryptografie (viz podkapitola 1.1.1),
- Asymetrická kryptografie (viz podkapitola 1.1.2).

1.1.1 Symetrická kryptografie

Tento způsob kryptografie je považován za základní zajištění důvěrnosti dat, které bylo vynalezeno už na počátku našeho letopočtu. Jejím hlavním předpokladem je využití stejného tzv. šifrovacího klíče pro funkce šifrování i dešifrování daného obsahu, jak je znázorněno na Obr. 1.1.

Problémem této kategorie je výměna šifrovacího klíče. Jelikož je šifrovací klíč pouze jeden pro obě operace, jeho odhalení umožňuje neautorizovanému subjektu získat a změnit data bez jakékoliv detekce.

Další komplikací je distribuce šifrovacího klíče mezi autorizovanými subjekty, která taktéž musí být zabezpečená dalšími kryptografickými prostředky.



Obr. 1.1: Model symetrické kryptografie.

V dnešní době slouží kryptografická primitiva využívající symetrických šifrovacích klíčů převážně k zabezpečení uchovávaných dat, přičemž jejich využití při komunikaci mezi systémy je často kombinováno s kryptografií asymetrickou (viz kapitola 1.1.2).

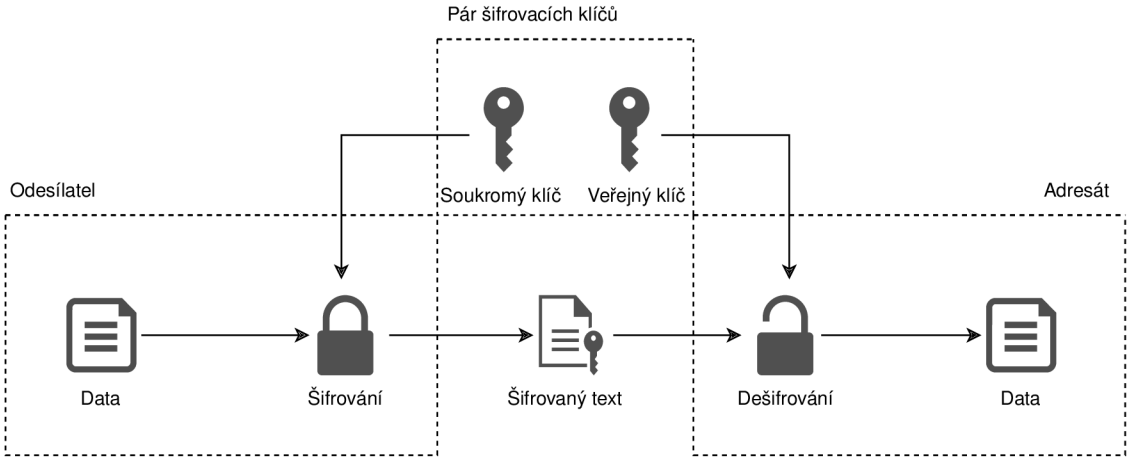
1.1.2 Asymetrická kryptografie

Hlavní odlišností oproti výše zmíněné symetrické kryptografii je v šifrovací klíči, který se zde využívá způsobem závislým na účelu celého algoritmu. Jeden z možných způsobu využití klíčů je znázorněn v Obr. 1.2. Šifrovací klíč zde tvoří pár (dvojice) klíčů. Veřejný klíč je obecně využíván k samotnému šifrování dat, k jejichž dešifrování je nutný klíč privátní (soukromý). Opačný způsob využití spočívá například při užívání digitálního podpisu popsaném v kapitole 1.3, kde je veřejný klíč užíván k ověření podpisu, zatímco privátní klíč slouží k podepisování.

Podstatou této skupiny kryptografických metod je výhradní vlastnictví privátního klíče subjektem přijímajícím data. Veřejný klíč je v tomto případě veřejně znám, a tudíž je možné zašifrovat jakákoliv data, avšak pouze majitel klíče privátního je schopen tato data dešifrovat.

1.2 Kryptografie s využitím eliptických křivek

Každá implementace kryptografie je tak bezpečná, jak bezpečný je její nejslabší článek. Všechny články této implementace kryptografického primitiva však slábnou postupem času z důvodu rostoucího výkonu výpočetních systémů doby. V praxi je tímto nejslabším článkem samotný šifrovací klíč. Tvůrci se snaží udržet daný algoritmus bezpečný postupným zvětšováním šifrovacího klíče, čímž znesnadňují útočníkovi jeho odhalení.



Obr. 1.2: Model asymetrické kryptografie.

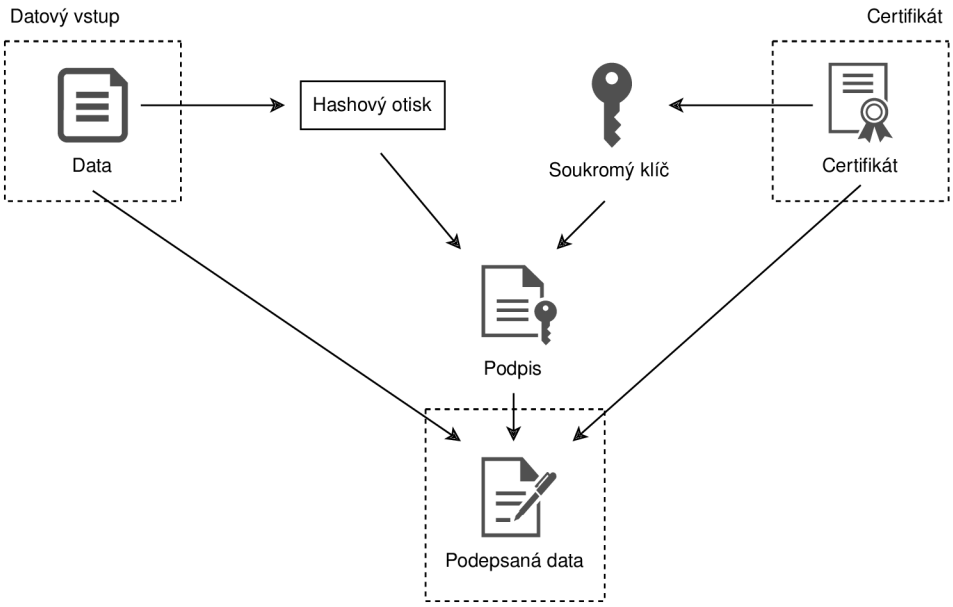
V některých případech se ovšem snaží tvůrci algoritmu velikost klíče snižovat pro jeho využití na méně výkonných platformách.

Řešením konfliktu mezi délkou klíče zajišťující požadovanou úroveň bezpečnosti a klíče s co nejmenší velikostí poskytují eliptické křivky. Jedná se o symetrické křivky kolem osy x, nahrazující šifrovací klíče pomocí jejich průsečíků s danou přímkou. Průsečíky této křivky a přímky dokážeme využít místo šifrovacích klíčů, čímž efektivně snížíme velikost klíčů nutných k prováděným operacím. Tímto způsobem dokážeme nahradit například 2048b klíč 256b klíčem vytvořeným za pomoci eliptických křivek při udržení stejné úrovně bezpečnosti. Detailní popis eliptických křivek však není předmětem této práce [1].

1.3 Digitální podpisy

Prokázání identity či stvrzení dat zajišťující nepopiratelnou autentizaci nemusí nutně znamenat vytvoření společně sdíleného tajemství. Svou identitu může subjekt prokázat také pomocí digitálních podpisů. Jedná se o datový řetězec přiložený ke zprávě (společně s certifikátem podpisu), či vyměněný mezi subjekty komunikace, který obsahuje všechny informace nutné k jednoznačnému prokázání identity. Tyto informace jsou především:

- Vlastník podpisu - osoba s výhradním vlastnictvím privátního klíče a certifikátu.
- Certifikační autorita - orgán vydávající certifikáty uživatelům na základě žádosti a ověření identity uživatele.
- Časové razítko - časový údaj označující přesný okamžik vytvoření podpisu.



Obr. 1.3: Model procesu podepisování dat.

Díky způsobu tvorby digitálního podpisu dokáže ověřovatel podpisu určit, kdy byla data podepsána, a zda s daty nemanipulovala nepověřená osoba [4]. Tento způsob se zakládá na nutnosti vytvoření otisku podepisovaných dat (viz Obr. 1.3), což zabraňuje záměně dat po jejich podepsání. Taková úprava obsahu by se při ověřování projevila na odlišném otisku dat v podpisu. Dále podpis zabezpečuje další z hlavních cílů kryptografie - nepopiratelnost na straně autora.

1.4 Hashovací funkce

Hashovací funkce jsou lehce vzdálenějším odvětvím kryptografie. Jejich primární funkcí je vytváření otisků sloužících k porovnávání dat bez znalosti jejich obsahu (viz Obr. 1.4). Příkladem využití hashovacích funkcí je ověřování přihlašovacích atributů, vytváření otisků dat pro digitální podpis, či vytvoření unikátního identifikátoru dat [5].

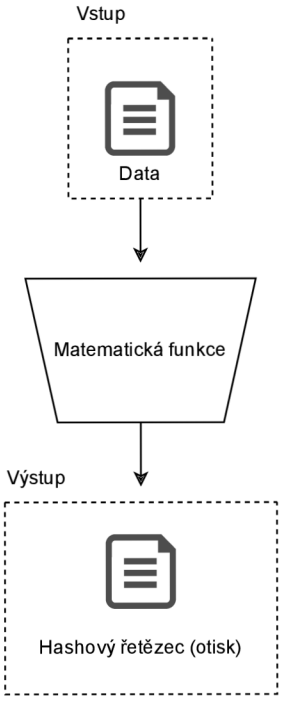
Tyto funkce jsou ze své podstaty jednosměrné, tedy není možné z hashového otisku obnovit původní data, ze kterých byl vytvořen. Tento vytvořený řetězec je vždy stejné délky, která je daná samotným algoritmem. Jedná se o poměrně rychlé algoritmy provádějící matematické operace s daty, na jejichž spolehlivosti však stojí velké množství kryptografických nástrojů.

Největším problémem hashovacích funkcí je nalezení tzv. kolizí.

- Kolize prvního řádu - nalezení dvou různých datových vstupů, jejichž výstupem je totožný otisk.

- Kolize druhého řádu - nalezení odlišného datového vstupu, jehož výstup bude totožný otisk jako u testovaného vstupu.
- Multikolize - nalezení skupin dat majících totožný hashový otisk.

Bezpečnost hashovací funkce je porovnávána na základě pravděpodobnosti tvorby těchto kolizí, složitosti nalezení původních dat z hashového otisku a odolnosti proti útokům na vnitřní strukturu funkce.



Obr. 1.4: Obecný model hashovací funkce.

2 Protokol DLMS/COSEM

Tato kapitola je věnována protokolu DLMS/COSEM [6], který je využíván systémy pro měření energií. Zkoumání průběhu protokolu a především užívání kryptografických primitiv je předmětem této diplomové práce. Tento protokol byl vybrán jako reprezentativní model protokolu využívaného v energetice.

2.1 Základní informace k protokolu

Device Language Message Specification (zkráceně DLMS) je protokol využívaný chytrými zařízeními pro autonomnější správu dat z oblasti energetiky. Byl vytvořen organizací DLMS UA založenou v roce 1997 na vývoj a podporu standardu výměny dat mezi chytrými zařízeními (převážně chytrými systémy určenými k měření hodnot).

Hlavní náplní protokolu DLMS je tvorba objektů a jejich překlad na zprávy, které je možné dále zasílat v komunikaci, společně s vytvářením a zabezpečením komunikace mezi entitami. Komunikace tohoto protokolu je založena na modelu klient-server, kde roli serveru hraje nejčastěji měřící zařízení a role klienta připadá sběrnému systému. Průběh komunikace začíná zasláním žádosti klienta na server, přičemž server odpoví odesláním požadovaných dat zpátky klientovi. Alternativním nastavením komunikace je automatické zasílání zpráv serverem klientovi dle nastavených podmínek bez nutnosti klientovy žádosti (např. periodické zasílání naměřených dat).

V případě této práce je zkoumán protokol DLMS ve spojení s objektovým modelem COSEM (Companion Specification for Energy Model), jež popisuje sémantiku jazyka. Vytvoření objektů COSEM umožňuje modelování pravidel energetického managementu a formální reprezentaci jednoduchých, či komplexních dat.

Další nutnou informací je implementace protokolu DLMS/COSEM na vrstvách v rámci modelu OSI/ISO. Zde se jedná o implementaci pravidel na prezenční vrstvu v rámci modelu COSEM a implementaci na vrstvy relační a transportní v rámci protokolu DLMS.

3 Zabezpečení protokolu DLMS/COSEM

Zajištění bezpečnosti je nutné v dnešní době při užívání jakéhokoliv protokolu a ani protokol DLMS/COSEM není výjimkou. V tomto případě využívá protokol řadu bezpečnostních mechanismů zajišťujících především autentizaci, autorizaci a zachování důvěrnosti.

3.1 Bezpečnostní mechanizmy

V této kapitole jsou detailněji popsány konkrétní bezpečnostní mechanismy podporované protokolem DLMS/COSEM [7]. Samotná implementace těchto mechanismů se však může lišit podle výrobce a při vzájemné komunikaci dvou systémů je nutné zabezpečit jejich kompatibilitu pro zachování funkčnosti a bezpečnosti.

Autentizace entit

Jedná se o potvrzení systému, že se skutečně jedná o entitu, za kterou se vydává. K ověření této identity dochází při snaze o vytvoření logického spojení klienta (tzv. Application Association) se serverem pomocí vzájemné výměny náhodných výzev a výsledků jejich kryptografického zpracování. V případě neúspěšné autentizace je zamezeno sestavení spojení [7].

Rozdělení přístupu na základě uživatelských práv

Rozdělení uživatelských práv je způsob zamezení přístupu uživatelů k datům v závislosti na roli uživatele. Každá role má stanovená pravidla, podle kterých může s daty nakládat - jedná se například o jejich zápis či čtení. Toto dělení má za cíl rozdělení pravidel hlavním uživatelem z důvodu ochrany vůči přístupu neautorizovaným osobám [7]. Výsledkem je poté soupis uživatelů s přidělenými rolemi, které by však uživatele neměly omezovat v rámci provádění určené činnosti.

Ochrana zpráv

Mechanismus ochrany zpráv probíhá na aplikační vrstvě, po jejímž průchodu jsou zprávy zasílány komunikační sítí. K ochraně zpráv dochází v závislosti na nastavených pravidlech:

- Zajištění důvěrnosti - použití šifrování.
- Zajištění integrity - použití autentizace.
- Zajištění přijmutí zprávy - použití elektronického podpisu.

Při vytváření Application Association se určují způsoby ochrany zprávy, které jsou poté uplatněny na všechny další zprávy zaslané ve vytvořené komunikaci [7].

Následně jsou veškeré ochranné mechanismy ze zprávy odstraňovány dle vrstvové hierarchie na straně klienta.

Ochrana dat

Ochrana dat je velmi podobná výše zmíněné ochraně zpráv, nicméně se zde jedná o ochranu obsahu samotných objektů vytvořených dle pravidel COSEM. Tato ochrana spočívá v objektu s názvem „Data protection“, v jehož atributu `protected_buffer` jsou chráněná data nejdříve napsána, a po aplikaci ochranných prostředků přidělena do patřičných atributů [7]. Obdobný proces v opačném pořadí se odehrává také na straně klienta, kde jsou data nejprve načtena do `protected_bufferu`, ze kterého následně klient data přečte.

Bezpečné aktualizace firmwaru

Protokol DLMS nabízí využití tzv. Image Transfer Mechanismu, který umožňuje klientovi nasazení nového chráněného obrazu systémového firmwaru na straně serveru. Tento obraz je následně před nasazením verifikován. Mechanismus nasazení firmwaru je implementován pomocí COSEM objektu obsahujícího veškeré atributy a metody k nahrání a aktivaci nového obrazu systému. Přenos těchto dat na server probíhá formou unicast či broadcast.

Ochrana komunikačních portů

Dalším bezpečnostním mechanismem jsou objekty typu „Communication port protection“. Jejich využitím může být ochrana proti škodlivým pokusům o navázaní spojení typu „brute force attack“ či „replay attack“. Toto zabezpečení je však v základní formě vzhledem k implementaci, kterou je pouze omezení počtu pokusů o navázání spojení [7].

Bezpečnostní záznamy

Bezpečnostní záznamy (angl. log) jsou jedním ze základních bezpečnostních mechanismů většiny informačních systémů. Jejich účelem je zaznamenávání aktivit systému pro zpětnou analýzu v případě selhání systému. V případě protokolu DLMS se jedná o monitorování chráněných zpráv a výměny dat implementovaných objekty „Profile Generic objects“ [7]. Jejich obsahem je záznam počtu správně či chybně zaslaných zpráv, výskyt errorů, nebo počet pokusů o vytvoření spojení. Tyto data jsou kdykoliv k dispozici k analýze a pro učinění preventivních, či reaktivních opatření.

3.2 Legislativní nařízení bezpečnostních algoritmů

Z důvodu zajištění alespoň minimální bezpečnosti komunikujících systémů jsou pravidla pro výběr kryptografických primitiv předepsány v podobě legislativních doporučení. Jedná se především o nastavení minimálního zabezpečení společně s doporučeními a úpravami primitiv odpovídajícími době vydání nařízení. Tato nařízení musejí být s postupem času aktualizována a při jejich tvorbě musí být brán ohled na schopnosti a výkon platforem, na které mají být implementována.

Mezi primární prameny legislativy kontrolující komunikaci energetických systémů jsou považována nařízení orgánů NIST a ENISA [8], [9]. K vytvoření obecných doporučení pro konfiguraci bezpečného zařízení je však nutné prostudování většího množství nařízení a následný výběr.

NIST

National Institute of Standards and Technology (zkráceně NIST [8]) je americkým orgánem shromažďujícím a opatrujícím technologické standardy pro systémy využívané ve Spojených státech Amerických. I přesto, že se jejich legislativa nevztahuje přímo na Českou republiku, jsou jejich nařízení brána jako doporučený standard pro zachování bezpečnosti v oblasti informačních systémů. V případě nařízení týkajících se bezpečnosti komunikačních systémů se obecně vyslovil tento úřad takto:

- „Přijatelný algoritmus je takový, který není znám jako nedůvěryhodný.“
- „Přijatelný šifrovací klíč je takový, jehož délka nemůže způsobit nedůvěryhodnost algoritmu samotného.“
- „Vybavení smart grid systémů musí mít průměrnou dobu životnosti alespoň dvaceti let.“

Algoritmy doporučené touto institucí v následujících kapitolách byly vybrány na základě oficiálních dokumentů instituce, a to konkrétně [10], [11], [12] a [13].

ENISA

European Union Agency for Cybersecurity (zkráceně ENISA [9]) je orgán Evropské Unie mající za cíl dosažení a standardizování vysoké úrovně kryptografické bezpečnosti výpočetních systémů napříč Evropou. Jeho prací je přispívání k celkové kryptografické politice, spolupráce s členskými státy při vytváření a zavádění vlastní legislativy v oblasti kryptografie, nebo také zvyšování důvěry v informační systémy a obecně příprava evropských systémů k vlastní kybernetické ochraně.

Sama organizace disponuje mnoha doporučeními ke správné konfiguraci systému pro splnění minimálních bezpečnostních předpokladů [15]. Jedno z doporučení týkajících se energetiky ohledně volby kryptografických parametrů je vytvoření dvou

schémat - jednu s kryptografickými prostředky dostatečnými pro zajištění bezpečnosti systému po dobu alespoň deseti let a druhou pro zajištění bezpečnosti po dobu třiceti až padesáti let.

Algoritmy doporučené touto institucí v následujících kapitolách byly vybrány na základě oficiálních dokumentů instituce, a to konkrétně [14] a [15].

NÚKIB

Národní ústav pro kybernetickou bezpečnost (zkráceně NÚKIB [16]) je ústředním správním orgánem pro kybernetickou bezpečnost informačních a komunikačních systémů na území České republiky. Dále má tento orgán za úkol informování veřejnosti o kybernetických nařízeních a dohledu nad jejich dodržováním. Jejich nařízení se týkají především doporučení typů algoritmů, které musejí být bezpečné alespoň ve střednědobém horizontu. Bezpečnost těchto nařízení se odvíjí od používaného algoritmu a délky klíčů.

NCSC

National Cyber Security Center (zkráceně NCSC [17]) je orgánem podporující systémy kritické infrastruktury Spojeného království Velké Británie a Severního Irska. Jedná se v podstatě o podobnou instituci jako NÚKIB operující v jiné zemi. Nařízení této instituce jsou v zásadě mírnější než ve výše zmíněných organizacích (případně totožná), takže jejich doporučení budou taktéž splněna v případě výběru algoritmů z tabulky následující kapitoly.

4 Kryptografická primitiva používaná v energetice

Jak již bylo zmíněno, veškeré informační systémy dnešní doby musejí splňovat alespoň minimální požadavky k zachování bezpečnosti komunikujících stran. Tyto požadavky se však liší účelem a místem, kde je systém užíván, a proto vytvoření obecného návrhu zabezpečení není zcela možný. Je však možné vytvořit seznam a popis nejčastěji využívaných kryptografických primitiv informačních systémů na základě doporučení předních institucí (NIST [8], ENISA [9], NÚKIB [16] a NCSC [17] v případě této diplomové práce), což je obsahem této kapitoly. K doplnění informací jsou následně primitiva krátce popsána společně s popisem jejich využití v reálných systémech.

4.1 Doporučení organizací

Některá doporučení vypsaná v tabulce jsou označená jako nedoporučená z důvodu nedostatečného dosavadního testování, nebo nedostatku provedených útoků, na jejichž základě je samotná bezpečnost hodnocena.

Tato myšlenka se týká převážně nařízení organizace NIST (například v případě blokových šifer [10]), která uznává jako hlavní šifrovací algoritmus AES, přičemž jiné blokové šifry nejsou doporučeny z důvodu jejich nepotřebnosti vzhledem ke standardu AES, jež zatím nebyl bezpečnostně zlomen.

V následujících podkapitolách jsou krátce popsány doporučované algoritmy rozdělené podle typu a základní informace o jejich principech. Součástí těchto popisů jsou tabulky s vyznačenými doporučeními jednotlivých institucí. Doporučení jsou zaznamenána v tabulce pomocí následujícího značení:

- Zelené značení - tento algoritmus je doporučený s danou délkou šifrovacího klíče.
- Žluté značení - jedná se o algoritmus označený v mezinárodní dokumentaci jako dosluhující (angl. legacy), tedy by neměl být používán v dlouhodobém měřítku.
- Červené značení - algoritmy takto označené jsou považovány za nevhodné či nedoporučené institucemi.
- Pomlčka - označení algoritmů, ke kterým nebylo nalezeno vyjádření dané instituce.

4.1.1 Blokové šifry

Blokové šifry jsou jedním z nejpoužívanějších typů symetrické kryptografie využívané téměř ve všech informačních zařízeních. Jedná se o základní způsob ochrany důvěrných dat při přenosu nebo jejich uložením. Svůj název dostaly po rozdělení dat určených pro šifrování do bloků, které jsou šifrovány samostatně. Mezi těmito bloky je v procesu šifrování přenášen odkaz předchozího bloku, aby byla zachována integrita dat. Tímto odkazem může být například využití části předešlého bloku jako inicializačního vektoru pro blok další.

Mezi nejznámější algoritmy této skupiny se nepochybně řadí algoritmy AES a jeho předchůdce 3DES. Kompletní seznam vyjádření institucí k bezpečnosti algoritmů blokových šifer je znázorněn v tabulce 4.1 níže. K blokovým šifrám se také pojí doporučení jejich provozních režimů zmíněné v tabulce 4.2.

AES

Advanced Encryption Standard (zkráceně AES) je šifrovací algoritmus od autorů Joana Daemena a Vincenta Rijmena (odtud původní název Rijndael), kteří s tímto algoritmem vyhráli veřejnou soutěž vyhlášenou institucí NIST v roce 1997 [18]. V roce 2001 byla tato šifra vyhlášena institucí NIST jako nejvhodnější návrh blokové šifry a díky jejím vlastnostem je široce využívána do dnešní doby. Jedná se o první šifru dostupnou veřejnosti, která byla zároveň uznána národní bezpečnostní agenturou NSA.

Šifra AES je rychlá softwarově i hardwarově a k dnešnímu datu není známý útok, při kterém by došlo k jejímu absolutnímu prolomení. Její podstatou je průběh určitého počtu „rund“ (cyklů složených se ze 4 stejných kroků), které tvoří:

- přidání podklíče,
- záměna bytů,
- prohození řádků,
- kombinování sloupců.

Šifrování pomocí AES je plně dostačující s ohledem na bezpečnost aktuální doby. Instituce NIST dokonce při svých doporučeních bere AES jako standard pro porovnávání výkonnosti, rychlosti a bezpečnosti nových šifer, což má za následek jejich vyjádření k ostatním blokovým šifrám jako nedoporučovaným, jelikož doposud nebyly podrobeny takovému testování, jako již dlouho užívaná šifra AES.

3DES

Triple Data Encryption Standard (zkráceně 3DES) je také symetrická bloková šifra, kterou lze doposud nalézt v reálných systémech. Tento algoritmus je využíván již od 70. let, přičemž byl převzat jako standard v roce 1977 organizací NIST. Její

podstatou je využití šifry DES, a to dokonce třikrát pro zaručení dostatečné ochrany dat.

Nutno dodat, že tato šifra je nyní již označena jako dosluhující a její nejbezpečnější podobu se třemi rozdílnými klíči už není doporučeno dlouhodobě využívat.

Tab. 4.1: Doporučené blokové šifry.

Blokové šifry (délka klíče)	NIST	ENISA	NÚKIB	Průnik doporučení
AES (128 bitů)	✓	✓	✓	✓
AES (192 bitů)	✓	✓	✓	✓
AES (256 bitů)	✓	✓	✓	✓
AES key-wrap (128 bitů)	✓	-	✓	✓
AES key-wrap (192 bitů)	✓	-	✓	✓
AES key-wrap (256 bitů)	✓	-	✓	✓
3DES (168 bitů)	✓	×	✓	×
3DES key-wrap (168 bitů)	✓	×	✓	×
Twofish (128 bitů)	×	✓	×	×
Twofish (192 bitů)	×	✓	✓	✓
Twofish (256 bitů)	×	✓	✓	✓
Serpent (128 bitů)	×	-	✓	×
Serpent (192 bitů)	×	-	✓	✓
Serpent (256 bitů)	×	-	✓	✓
Camellia (128 bitů)	×	✓	✓	×
Camellia (192 bitů)	×	✓	✓	✓
Camellia (256 bitů)	×	✓	✓	✓

Tab. 4.2: Doporučené provozní režimy blokových šifer.

Provozní režimy blokových šifer	NIST	ENISA	NÚKIB	Průnik doporučení
GCM	✓	✓	✓	✓
CCM	✓	✓	✓	✓
Režimy schématu Encrypt-then-MAC				
ECB	×	×	×	×
CBC	✓	✓	✓	×
CFB	✓	✓	✓	✓
OFB	✓	✓	✓	✓
CTR	✓	✓	✓	✓

4.1.2 Proudové šifry

Tento typ symetrických šifer využívá principu kombinace vstupního datového toku s pseudonáhodným řetězcem vystupujícím jako základní šifrovací prvek.

Důležitým obsahem proudové šifry je pseudonáhodná funkce mající za vstup klíč, na jehož základě funkce následně vytvoří řetězec. Datový vstup je kombinován s takto vytvořeným řetězcem nejčastěji pomocí funkce XOR, čímž vzniká jako výstup šifrovaný datový řetězec [19]. K dešifrování je poté nutné znát vstup pro pseudonáhodnou funkci generování řetězce.

Jejich využití je v praxi uplatňováno na výpočetní systémy se slabším hardwarem, jelikož ke svému provozu nepotřebují velkou výpočetní sílu. Nevýhodou použití těchto šifer je potencionálně větší náchylnost ke kryptoanalitickým útokům v případě nedokonalé implementace.

Algoritmy doporučené institucemi jsou zaznamenány v tabulce 4.3 níže. Při doporučení proudové šifry ChaCha20 se jedná o používání šifry s funkcí Poly1305, určenou k ověřování integrity dat a autentizaci zprávy.

Tab. 4.3: Doporučené proudové šifry.

Proudové šifry (délka klíče)	NIST	ENISA	NÚKIB	Průnik doporučení
SNOW 2.0 (128 bitů)	-	✓	✓	×
SNOW 2.0 (256 bitů)	-	✓	✓	✓
SNOW 3G (128 bitů)	-	✓	✓	✓
SNOW 3G (256 bitů)	-	✓	✓	✓
ChaCha20 (256 bitů)	-	-	✓	✓

4.1.3 Hashovací funkce

Hashovací funkce působí v systémech jako matematické funkce měnící datový vstup na datový výstup o konstantní délce závislé na zvolené hashovací funkci (viz Obr. 1.4) [5]. Důležitou vlastností je jednocestnost těchto funkcí, tzn. že není možné získat datový vstup na základě výstupních dat.

Využití hashových otisků je dnes přítomné téměř v každém informačním systému. Například algoritmy podpisů a autentizací spoléhají na hashové otisky jako reprezentaci dat.

Nejznámějšími užívanými hashovacími funkcemi jsou funkce rodiny SHA (angl. Secure Hash Algorithm). Existují však hashovací algoritmy, které nejsou vyhovujícími k používání z důvodu tvoření velkého množství kolizí při tvorbě výstupních dat, či nalezení způsobu porušení jednocestnosti funkce.

Algoritmy doporučené institucemi jsou zaznamenány v tabulce 4.4 níže.

Tab. 4.4: Doporučené hashovací algoritmy.

Hashovací algoritmy (délka výstupu)	NIST	ENISA	NÚKIB	Průnik doporučení
SHA-1	×	✓	×	×
SHA-2 (256 bitů)	✓	✓	✓	✓
SHA-2 (384 bitů)	✓	✓	✓	✓
SHA-2 (512 bitů)	✓	✓	✓	✓
SHA-3 (256 bitů)	✓	✓	✓	✓
SHA-3 (384 bitů)	✓	✓	✓	✓
SHA-3 (512 bitů)	✓	✓	✓	✓

4.1.4 Algoritmy distribuce klíčů

Způsob distribuce klíčů je jedním ze zásadních problémů užívání asymetrických šifer. K řešení tohoto problému jsou proto implementovány další algoritmy umožňující ustanovení klíčů skrze nezabezpečený kanál.

Protokol Diffie-Hellman (a jeho varianta s využitím eliptických křivek; zkráceně DH) je považován za základní protokol umožňující tuto distribuci. Průběh protokolu je založen na modulární aritmetice, jehož výstupem je výpočet společného šifrovacího klíče při znalosti veřejných parametrů ve spojení s vyměněnými daty v průběhu komunikace.

Algoritmy zmíněné v tabulce s příponou „KEM“ (angl. Key Encapsulation Mechanism) využívají myšlenky algoritmu DH, kterou následně obohacují o další bezpečnostní prostředky s využitím eliptických křivek.

Algoritmy doporučené institucemi jsou zaznamenány v tabulce 4.5 níže.

4.1.5 Algoritmy digitálního podpisu

Algoritmy digitálního podpisu umožňují vytvoření specifických dat nahrazujících klasický vlastnoruční podpis v informačních systémech [4]. K vytvoření podpisu využívají soukromých klíčů vydaných certifikačními autoritami a hashových otisků podepisovaných dat vytvořených hashovými algoritmy systému [5].

Obdobně jako algoritmy distribuce klíčů, také tyto algoritmy umožňují využití eliptických křivek. Dle názorů institucí je doporučené v případě podpisů implementace algoritmů s těmito eliptickými křivkami, případně základních typů algoritmů s klíči o minimální délce 3072 bitů.

Algoritmy doporučené institucemi jsou zaznamenány v tabulce 4.6 níže.

Tab. 4.5: Doporučené algoritmy distribuce klíčů.

Algoritmy distribuce klíčů (délka klíče)	NIST	ENISA	NÚKIB	Průnik doporučení
DH (2048 bitů)	✓	-	✓	×
DH (3072 bitů)	✓	-	✓	✓
ECDH (224 bitů)	✓	-	✓	×
ECDH (256 bitů)	✓	-	✓	✓
ECIES-KEM (224 bitů)	-	✓	✓	×
ECIES-KEM (256 bitů)	-	✓	✓	✓
PSEC-KEM (224 bitů)	-	✓	✓	×
PSEC-KEM (256 bitů)	-	✓	✓	✓
ACE-KEM (224 bitů)	-	-	✓	×
ACE-KEM (256 bitů)	-	-	✓	✓
RSA-KEM (2048 bitů)	×	✓	✓	×
RSA-KEM (2048 bitů)	✓	✓	✓	✓

Tab. 4.6: Doporučené algoritmy digitálního podpisu.

Algoritmy digitálního podpisu (délka klíče)	NIST	ENISA	NÚKIB	Průnik doporučení
DSA (2048 bitů)	×	✓	×	×
DSA (3072 bitů)	✓	✓	✓	✓
ECDSA (256 bitů)	✓	✓	✓	✓
RSA (2048 bitů)	×	✓	×	×
RSA (3072 bitů)	✓	✓	✓	✓
EC-Schnorr (224 bitů)	-	✓	✓	×
EC-Schnorr (256 bitů)	-	✓	✓	✓

4.1.6 Generátory náhodných čísel

Tato sekce se netýká přímo algoritmů zajišťující kryptografickou bezpečnost, avšak funkce generování náhodných čísel jsou využívány bezpečnostními mechanismy, a tudíž musejí taktéž obstát v určité bezpečnostní úrovni [20]. Tato bezpečnost je hodnocena na základě rozložení vygenerovaných čísel.

Generátory využívané v informačních systémech jsou většinou pseudonáhodné, jelikož jejich generování není založené na náhodných fyzikálních procesech, ale je čistě softwarové. Toto generování může také obsahovat vstupy (tzv. „seed“), díky nimž je možné ovlivnit vygenerovaný výstup.

Správně operující generátor náhodných čísel je základem mnoha kryptografických operací dnešní doby a vyjádření institucí k jejich bezpečnosti je znázorněno v tabulce 4.7 níže.

Tab. 4.7: Doporučené generátory náhodných čísel.

Generátory náhodných čísel	NIST	ENISA	NÚKIB	Průnik doporučení
Hash_DRBG	✓	-	-	✓
HMAC_DRBG	✓	-	-	✓
CTR_DRBG - 3DES	✓	-	-	✓
CTR_DRBG - AES	✓	-	-	✓

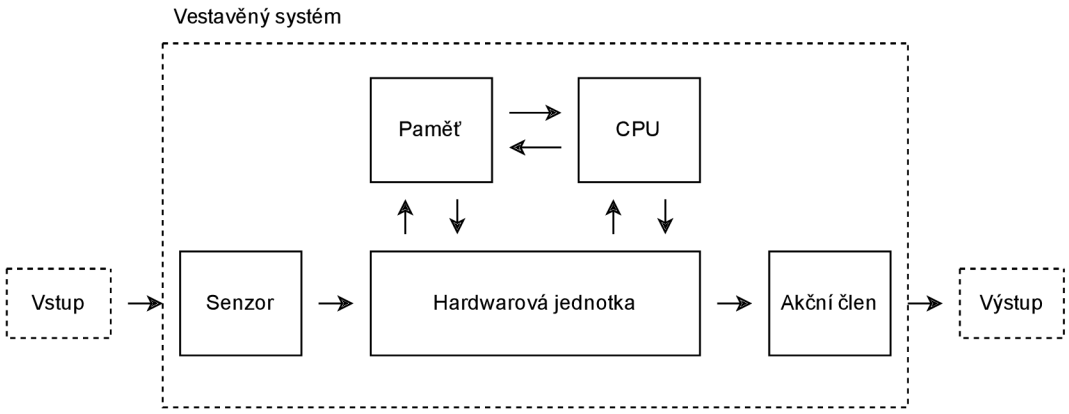
5 Embedded systémy

Přínosem praktické části diplomové práce je testování kryptografických primitiv na embedded platformách, k čemuž je nutný jejich správný výběr. Obsahem této kapitoly je tedy popis základního principu embedded platforem společně s výčtem vhodných kandidátů na testování.

5.1 Využití embedded systémů

Vestavěný systém (také zabudovaný systém, či embedded system) je jednoúčelový počítač, sestavený především z procesoru, paměti a vstupních či výstupních periferií mající přiřazenou úlohu, kterou vykonávají při svém provozu [21]. Tyto jednoúčelové počítače mohou fungovat samostatně při zpracovávání jednodušších úloh, nebo být součástí většího systémového celku, v němž zastávají pouze svou roli, pro kterou byly naprogramovány.

Základem těchto systémů jsou především mikrokontroléry a paměti společně s integrovanými obvody uzpůsobenými k přenosu výpočtů v reálném čase. Složitost systému, stejně jako softwarová nadstavba, závisí plně na jejich určenému využití (viz Obr. 5.1).



Obr. 5.1: Obecný model embedded platformy.

V současnosti jsou tyto systémy hojně využívány od nejrůznějších domácích spotřebičů až po základní prvky automatizací informačních soustav.

Mezi nejznámější embedded platformy se nepochybně řadí výrobky britské společnosti Raspberry Pi [22]. Produkty této firmy budou předmětem testování v navazující praktické části z důvodu univerzálnosti těchto systémů. Výsledné testování kryptografických primitiv na těchto platformách bude sloužit jako předpoklad při jejich případném využití v praxi v oblasti energetiky.

Při výběru platforem vhodných k testování bylo také přihlédnuto k využití platforem s hardwarovou akcelerací. Za hardwarovou akceleraci je považována část výpočetního systému vykonávající specifické výpočetní úkony. Implementace této akcelerace způsobuje zrychlení celého systému díky efektivnějšímu rozložení výpočetní zátěže. Bohužel však žádná platforma s hardwarovou akcelerací nebyla shledána vhodnou pro testování prováděné v praktické části diplomové práce.

5.2 Vybrané embedded platformy

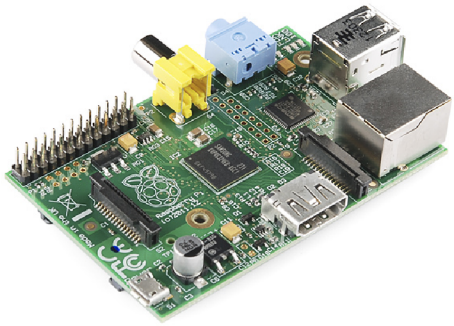
Následující sekce se věnují alespoň krátkému popisu výrobků vybraných jako vhodné platformy pro testování [23].

5.2.1 Raspberry Pi B

První vybranou platformou je Raspberry Pi B (viz Obr. 5.2) [24]. Jedná se o předposlední produkt původní řady Raspberry Pi osazený čipem Broadcom BCM2835. Konkrétní hardwarové parametry jsou sepsány v tabulce [5.1 níže.

Tab. 5.1: Parametry zařízení Raspberry Pi B+ 512RAM.

Architektura	ARMv6Z (32-bit)
SoC	Broadcom BCM2835
CPU	jednojádrové ARM1176JZF-S @ 700MHz
GPU	Broadcom VideoCore IV @ 250 MHz, OpenGL ES 2.0, MPEG-2 and VC-1 (s licencí), 1080p30 H.264/MPEG-4 AVC high-profile decoder and encoder
Paměť (SDRAM)	512 MB (sdílená s GPU)
Vstupy/Výstupy	2x USB 2.0 1x MIPI 1x HDMI (1.3) 1x 3.5 mm phone jack 1x I2C Bus
Interní paměť	SD, MMC, slot SDIO karty
Integrovaná síť	10/100 Mbit/s Ethernet (8P8C), USB adaptér přes USB hub
Jmenovitý výkon	200 mA (1 W) v průměru na volnoběh, maximálně 350 mA (1,75 W) při zatížení periferiemi



Obr. 5.2: Ukázka zařízení Raspberry Pi B.

5.2.2 Raspberry Pi 2

Raspberry Pi 2 Model B (viz Obr. 5.3) je zástupcem 2. generace řady Raspberry Pi, jež nahradil předchozí model Raspberry Pi B. Jeho hlavní předností oproti jeho předchůdci je novější CPU s větším počtem jader, a tedy větší výpočetní silou. Konkrétní hardwarové parametry jsou sepsány v tabulce 5.2 níže.

Tab. 5.2: Parametry zařízení Raspberry Pi 2 Model B.

Architektura	ARMv7 (32-bit)
SoC	Broadcom BCM2836
CPU	Čtyřjádrový ARM Cortex-A53 @ 900Mhz
GPU	Dual Core VideoCore IV @ 250 MHz, OpenGL ES 2.0, hardwarová akcelerace OpenVG, 1080p30 H.264
Paměť (SDRAM)	1GB LPDDR2
Vstupy/Výstupy	4x USB 2.0 1x MIPI 1x HDMI (1.3 & 1.4) 1x 3.5 mm jack 1x I2C 1x DSI
Interní paměť	MicroSDHC slot
Integrovaná síť	10/100 Mbit/s Ethernet (8P8C), adaptér na USB hubu (3,4)
Jmenovitý výkon	800 mA (4,0 W)

Obr. 5.3: Ukázka zařízení Raspberry Pi 2 Model B.

5.2.3 Raspberry Pi 3

Model Raspberry Pi 3 (viz Obr. 5.4) je prvním modelem osazeným 64bitovým CPU, který by dle výroku výrobce měl zvýšit výkonnost přístroje až o 50%. Mezi další novinky tohoto modelu patří také implementace Bluetooth a WiFi modulů. Konkrétní hardwarové parametry jsou sepsány v tabulce 5.3 níže.

Obr. 5.4: Ukázka zařízení Raspberry Pi 3 Model B.

Tab. 5.3: Parametry zařízení Raspberry Pi 3 Model B.

Architektura	ARMv8-A (64/32-bit)
SoC	Broadcom BCM2837B0
CPU	64-bit čtyřjádrový ARM Cortex-A53 @ 1.4GHz
GPU	Dual Core VideoCore IV @ 250 MHz, OpenGL ES 2.0, MPEG-2 a VC-1 (s licencí) 1080p30 H.264
Paměť (SDRAM)	1GB (sdílená s GPU)
Vstupy/Výstupy	4x USB 2.0 1x MIPI 1x HDMI (1.3 & 1.4) 1x 3.5 mm jack 1x I2C 1x DSI
Interní paměť	MicroSDHC, USB Boot Mode
Integrovaná síť	Gigabitový Ethernet skrze USB 2.0 + WiFi 802.11ac, Bluetooth 4.2 BLEl, možnost PoE napajení
Jmenovitý výkon	300 mA (1,5 W) v průměru na volnoběh, 1,34 A (6,7 W) při maximálním vytížení periferiemi

5.2.4 Raspberry Pi 4

Raspberry Pi 4 (viz Obr. 5.5) patří k posledním vydaným modelům firmy Raspberry Pi. Hlavními přínosy tohoto modelu jsou zvětšení paměti, rozšíření počtu portů a jejich inovace, přidání micro-HDMI a nový konektor napájení USB-C. Konkrétní hardwarové parametry jsou sepsány v tabulce 5.4 níže.

Tab. 5.4: Parametry zařízení Raspberry Pi 4.

Architektura	ARMv8 (64-bit)
SoC	Broadcom BCM2711
CPU	64-bit čtyřjádrový ARM Cortex-A72 @ 1.5GHz
GPU	Dual Core VideoCore IV @ 250 MHz, OpenGL ES 3.1, Vulkan, 1080p30 H.264 4Kp60 H.265
Paměť (SDRAM)	2 GB / 4 GB / 8 GB LPDDR4
Vstupy/Výstupy	2x USB 2.0 2x USB 3.0 1x HDMI 1x 3.5 mm jack 1x I2C 1x MIPI DSI 1x MIPI CSI
Interní paměť	MicroSDHC, USB Boot Mode
Integrovaná síť	Gigabitový Ethernet, Bluetooth 5.0, 2.4GHz a 5GHz IEEE 802.11.b/g/n/ac Wi-Fi
Jmenovitý výkon	3A (15W)

Obr. 5.5: Ukázka zařízení Raspberry Pi 4 Model B.

6 Rešerše softwarových nástrojů pro testování embedded platforem

Knihovny jsou nedílnou součástí jakéhokoliv softwarového nástroje informačních zařízení. Jejich využití je rozsáhlé a výběr knihovny záleží na funkcích požadovaných od výpočetního systému. Tato kapitola se zaobírá studií právě těchto knihoven doplňujících systémy o operace s kryptografickými primitivy.

6.1 OpenSSL

OpenSSL je jednou z nejznámějších používaných kryptografických knihoven. Jejím hlavním zaměřením je implementace protokolu SSL, který umožňuje zabezpečení komunikace šifrováním a autentizací komunikujících stran [27]. Tato knihovna je volně dostupná a ve svém obsahu nabízí širokou škálu kryptografických algoritmů s různými vstupními parametry. Zároveň je knihovna OpenSSL podporována většinou nejužívanějších operačních systémů, jako například Windows, MacOS, Linux, nebo FreeBSD. Mezi funkce nabízené touto knihovnou patří:

- Blokové šifry - AES (módy GCM, CCM, CBC), 3DES (s klíčem o délce 168 bitů) a Camellia s variabilními délkami klíčů (128 a 256).
- Proudové šifry - ChaCha-Poly1305.
- Algoritmy digitálních podpisů - RSA, DSA a ECDSA.
- Hashovací algoritmy - SHA1, všechny algoritmy rodin SHA2 a SHA3.

Jedním z důležitých nástrojů, s nímž se bude dále pracovat, je modul `openssl speed`, který slouží k testování kryptografických algoritmů z časového hlediska. Tento modul je však možné využít pouze u některých kryptografický algoritmů, a proto v případě testování všech zmíněných algoritmů je třeba využití jiné alternativy.

6.2 Crypto++

Jedná se o volně dostupnou open-source knihovnu psanou v jazyce C++ autorem Wei Dai [28]. Je hojně využívána v oblastech akademických až po sféru obchodní. Podpora této knihovny je také rozsáhlá (obdobně jako tomu bylo u OpenSSL), jmenovitě na operačních systémech Windows, Linux, MacOS a BSD systémech. Výhodou oproti OpenSSL je zaměření pouze na implementaci kryptografických primitiv, včetně těch méně používaných či známých. Mezi tyto méně známé algoritmy implementované knihovnou Crypto++ jsou například:

- Blokové šifry - Twofish, Serpent nebo Blowfish včetně všech verzí s různými klíči.

- Podporované módy blokových šifer - ECB, CBC, CFB, OFB a CTR.
- Hashovací funkce - SHAKE a WHIRLPOOL.
- Algoritmy digitálního podpisu - ElGamal, Rabin-Williams algorithm, nebo ECGDSA (německá verze algoritmu DSA na eliptických křivkách).

Tato knihovna také disponuje implementací knihoven považovaných za zastaralé z důvodu zpětné kompatibility a jejich historického významu. Těmito šiframi jsou například hashovací algoritmy rodiny MD (například nejznámější MD-5) nebo původní verze šifry DES.

6.3 WolfCrypt

WolfCrypt engine je odlehčená knihovna napsaná v jazyce ANSI C určená právě za cílem implementace kryptografický nástrojů na embedded platformy, či jinak výkonově limitované platformy. Zároveň je součástí většího projektu WolfSSL. Knihovna je k tomuto účelu také vyvíjena, a je tedy menší než ostatní knihovny, přičemž jejími hlavními přednostmi jsou rychlost společně s kryptografickou výbavou. Vývojáři se také pyšní implementacemi moderních šifer, jako například RABBIT, HC-128 a NTRU. Tato knihovna je také zahrnuta ve standardech organizace NIST, a to konkrétně v nařízení FIPS 140-2 a FIPS 140-3.

Co se týče kryptografické výbavy, tak kromě výše zmíněných moderních šifer nepředstavuje tato knihovna v zásadě nic nového oproti konkurenci kromě kompatibility s vrstvou OpenSSL [29].

6.4 Libgcrypt

Libgcrypt je veřejná kryptografická knihovna původně založená na zdrojovém kódu GnuPG, což je svobodná implementace standardu OpenPGP, který je využíván pro kryptografická zabezpečení emailové korespondence [30]. K jejímu vývoji se využívá služby GIT, kde jsou změny sledovány a schvalovány hlavními tvůrci této knihovny. Předností této knihovny jsou vlastní implementace výpočtů, které jsou limitovány paměťovými vlastnostmi systému (také nazvané „Výpočty s libovolnou přesností“). Implementace těchto výpočtů jdou ruku v ruce s vlastním zdrojovým kódem pro tyto operace psané v jazyce Assembler, které jsou upraveny pro různé procesory.

Dále knihovna disponuje funkcí sledování a počítání entropie pro výpočetní jednotky využívající operační systémy založené na UNIX nebo Windows.

Z hledisky výbavy kryptografickými primitivy má tato knihovna k dispozici obdobně rozsáhlou sbírku algoritmů, jako výše zmíněná knihovna Crypto++.

6.5 LibreSSL

Knihovna LibreSSL je svobodnou implementací protokolu TLS (společně s jeho předchůdcem SSL), založenou vývojáři operačního systému OpenBSD jako samostatnou větev projektu OpenSSL [31]. Podpora LibreSSL se nachází hlavně na operačním systému OpenBSD, společně s podporou na systémech Linux, MacOS a FreeBSD. Tato knihovna však disponuje menším počtem kryptografických primitiv než její konkurenti, a i přes snahu vývojářů vyvarovat se chybám, jež se vyskytovaly v knihovně OpenSSL, jedná se stále o slabší variantu knihovny OpenSSL. V této práci je knihovna LibreSSL zmíněna za účelem nabídnutí alternativy ke známějším knihovnám.

7 Testování platformy Raspberry Pi 3

Obsahem praktické části této práce je testování kryptografických primitiv na zvolené embedded platformě. V tomto případě bylo vybráno zařízení Raspberry Pi 3 zmíněné v podkapitole 5.2.3. Výsledky testování byly z příkazové řádky zaznamenány do textového souboru pro následnou analýzu. Tyto záznamy testování šifer hojně užívaných v praxi jsou zaznamenány v tabulkách níže.

Testování probíhalo za pomoci knihovny OpenSSL a příkazu `openssl speed` (zmíněno v sekci [6.1]), který slouží k testování celkového času provedení kryptografické operace s různými velikostmi vstupů a následný výpočet teoretické rychlosti algoritmu. Tento příkaz testoval algoritmy:

- Blokových šifer - AES, DES, IDEA, SEED, Blowfish, Cast, Camellia, RC5 (viz Tab. 7.1),
- Proudových šifer - RC4,
- Hashovacích algoritmů - MD4, MD5, SHA1, SHA256, SHA512 a Whirlpool (viz Tab. 7.2),
- Podpisových algoritmů - RSA, DSA, ECDSA a EdDSA (viz Tab. 7.3),
- Algoritmů distribuce klíče - ECDH (viz Tab. 7.4),
- Algoritmů autentizačního kódu zprávy - HMAC (s hashovací funkcí MD5).

7.1 Výsledky testování

V následujících tabulkách je zaznamenána rychlost převodu vstupu (o velikosti 16B až 16384B) na výstup daného algoritmu v jednotkách kB/s. Níže zmíněné algoritmy byly testovány pouze v módu CBC.

Tab. 7.1: Výsledky testování blokových šifer na Raspberry Pi 3.

Type	16 B	64 B	256 B	1024 B	8192 B	16384 B
DES	17053,96	18324,80	18731,61	18900,85	18868,91	18863,45
TDES	6476,90	6664,51	6720,00	6763,88	6742,02	6739,29
AES-128	38161,39	45332,25	47877,89	48661,92	48739,67	48698,71
AES-192	33351,06	38834,41	40851,61	41147,05	41301,33	41431,25
AES-256	30208,93	30208,93	36294,98	36542,81	36631,89	36740,71
Camellia-128	29999,11	34678,69	36040,02	36458,50	36563,63	36558,17
Camellia-192	24426,39	27499,30	28316,50	28561,75	28641,96	28617,39
Camellia-256	24464,38	27390,40	28414,20	28559,02	28622,85	28713,10

Dle výsledků obsažených v tabulce 7.1 je algoritmus AES nejvhodnější volbou ze zmíněných blokových šifer. I s velikostí klíče 256b stále dosahuje obdobné rych-

losti jako konkurenční šifry, avšak s poloviční velikostí klíče. Z tabulky lze také porovnat šifru AES se svým předchůdcem, šifrou DES (následně TDES, či 3DES), která dosahuje polovičních rychlostí. V případě algoritmu TDES se jedná o rychlost šestinovou vůči algoritmu AES. Test také obsahuje méně užívané algoritmy, jako například SEED, Cast, nebo RC5. Algoritmy blokových šifer byly dle dosažených výsledků velmi dobře optimalizovány.

Tab. 7.2: Výsledky testování hashovacích algoritmů na Raspberry Pi 3.

Type	16 B	64 B	256 B	1024 B	8192 B	16384 B
SHA1	19477,28	52320,60	105300,42	139922,77	155364,01	156543,66
SHA256	12779,69	31841,28	59578,20	77359,95	84314,79	84885,50
SHA512	7391,85	29340,90	46682,20	66106,37	75678,74	76174,68

V případě hashovacích algoritmů se také jedná o dobře optimalizované algoritmy, kdy algoritmus SHA1 je schopen dosahovat rychlostí přes 150 MB/s (viz Tab. 7.2). Rychlosti uvedené v této tabulce jsou v jednotkách kB/s.

Nedostatkem tohoto obecného testování je však absence zastoupení rodiny hashovacích funkcí SHA-3. Implementace těchto funkcí může být obsahem následujících aktualizací algoritmů modulu OpenSSL speed.

Tab. 7.3: Výsledky testování podpisových algoritmů na Raspberry Pi 3.

Type	sign (s)	verify (s)	sign/s	verify/s
RSA-2048	0,011976	0,000269	83,5	3723,7
RSA-3072	0,032961	0,000553	30,3	1808,8
DSA-2048	0,003584	0,003000	279,0	333,3
ECDSA-256	0,0004	0,0013	2252,3	745,4

Výsledky algoritmů digitálních podpisů se skládají z časových údajů nutných k podepsání a ověření dat společně s přepočtem množství podpisů a ověření, které je algoritmus schopen provést za sekundu. Z tabulky 7.3 je patrný markantní rozdíl časové náročnosti mezi původními verzemi podpisových algoritmů a verzemi s využitím eliptických křivek.

Zajímavým výsledkem v této kategorii je rychlost ověření, které je algoritmus RSA schopen v porovnání s rychlostí podepisování dat.

Algoritmy výměny klíčů byly testovány pouze ve verzi využívající eliptické křivky. Těchto verzí je však testováno hned několik a každá s jinou vstupní eliptickou křivkou (převážně se jedná o křivky *nist* a *brainpool*). Algoritmy zmíněné v tabulce 7.4 jsou vybrány jako nejužívanější a patří mezi nejrychlejší z celého výběru. Číselná hodnota

výsledku v tomto případě znamená čas vykonání operace a následný převod na počet operací za sekundu.

Tab. 7.4: Výsledky testování algoritmů výměny klíčů na Raspberry Pi 3.

Type	op (s)	op/s
ECDH (nistp224)	0,0046	217,2
ECDH (nistp256)	0,0010	1040,0

7.2 Závěr testu

Dosažené výsledky odpovídají reálným předpokladům vycházejícím ze složitosti a náročnosti kryptografických primitiv implementovaných v knihovně OpenSSL. Výpis výsledků je stručný, přehledný a obsahuje nejnutnější informace spojené s testováním rychlosti daných primitiv.

Nevýhodami testu mohou být například absence členění primitiv (jedná se o výpis pod sebou), malý počet variant testovaných šifer (příkladem je testování šifry AES pouze v módech CBC a IDE), případně absence větší struktury testu (šifry mající tentýž účel mohou být vypsány jak na začátku, tak na konci).

Naopak výhodami tohoto testu jsou především rychlost, jednoduchost a přesnost. Díky těmto vlastnostem je možné použít tento test na všech operačních systémech, a tedy na téměř jakémkoliv zařízení.

8 Návrh testovacího nástroje

Cílem praktické části této diplomové práce je vytvoření automatizovaného nástroje pro testování kryptografických primitiv na embedded platformách. Vytvořený nástroj by měl být schopen určitým způsobem prezentovat výsledky daného testování.

8.1 Teorie nástroje

Základní částí automatizovaného testovacího nástroje jsou implementace testovaných algoritmů jako takových. Velkou pomocí jsou v tomto případě kryptografické knihovny (viz kapitola 6).

Kryptografické knihovny obsahují implementace velkého množství kryptografických primitiv, avšak ve valné většině případů spoléhají především na implementace frekventovaně užívaných algoritmů. Při tvorbě testovacího nástroje však může nastat situace, kdy je požadovaná implementace kryptografického primitiva zastaralá, či dokonce ani v dané konfiguraci neexistuje z důvodu upřednostnění konkurenční šifry širší veřejností z praktických nebo bezpečnostních důvodů. V takovém případě je nutné vyhledat vlastní implementaci samotného algoritmu, která je alespoň v minimální míře otestovaná a bezpečná. Užívání vlastní implementace však není doporučené pro komerční účely, jelikož optimalizace algoritmu bývá daleko nižší než je tomu v případě algoritmu implementovaného v knihovně a její bezpečnost nemůže být dostatečně prokázána. Testování takto implementovaného algoritmu se také výrazně odráží na spotřebě zdrojů (paměť, výkon, apod.).

Nadřazenější částí testovacího nástroje je postup testování jednotlivých algoritmů. V případě tohoto nástroje se jedná o postupné testování algoritmu s různými parametry, zaznamenání naměřených výsledků následované testováním dalšího algoritmu s parametry jinými. Tento proces je opakován dokud není provedeno testování posledního algoritmu v předem definovaném seznamu (znázorněno v diagramu na Obr. 8.1). Při testování těchto algoritmů by měly být implementovány nástroje pro měření podstatných údajů, které mohou následně sloužit k porovnání dosažených výsledků. Mezi hlavní sledované charakteristiky algoritmů by měly patřit především:

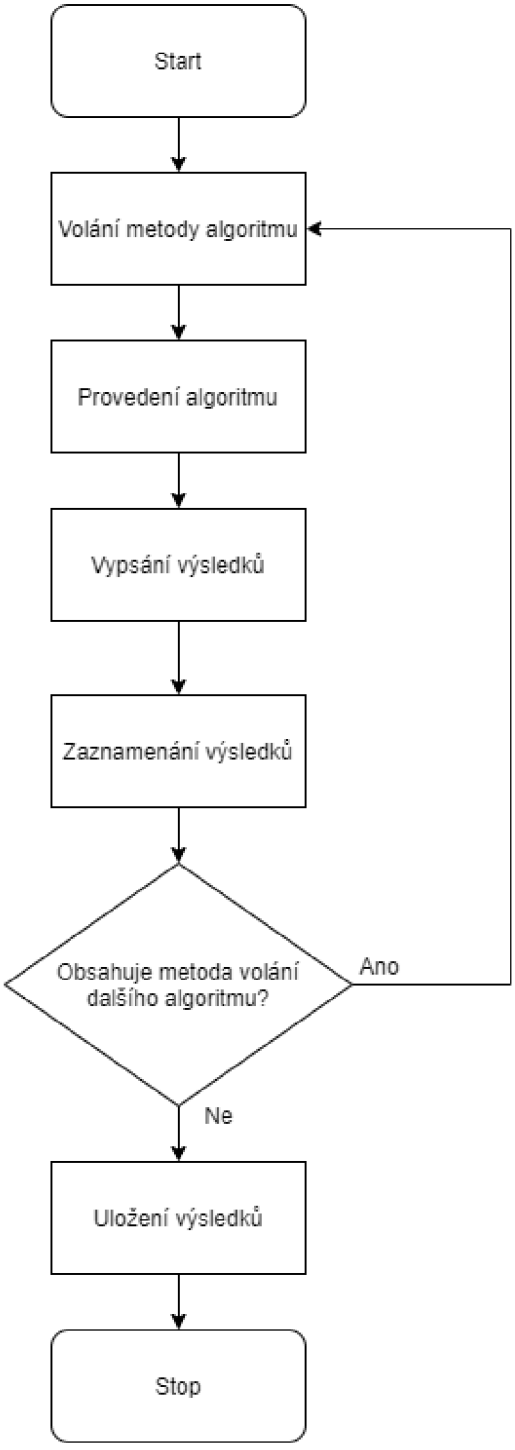
- čas využitý procesorem ke zpracování instrukcí programu,
- rychlost zpracování instrukcí algoritmu za danou jednotku času,
- využití paměti při zpracování instrukcí,
- a spotřeba energie.

Dalším krokem nástroje je ukládání naměřených výsledků do formátu, který je možné později vyvolat k porovnání. Tato data musí být uložena v podobě uzpůsobené pro následné zpracování programem. Při těchto podmínkách přicházejí v úvahu

techniky ukládání dat do souborů formátu CSV, JSON nebo XML. Data takto uložená je možné relativně jednoduše převést zpátky do podoby, ve které mohou být dále zpracovávána.

Posledním krokem je prezentace výsledků. V případě konzolového nástroje je možné se inspirovat nástrojem OpenSSL `speed`, který své výsledky vypisuje do konzole po zadání příkazu pro spuštění nástroje. Uživatelsky přijatelnější verze by však byla vytvoření grafického uživatelského rozhraní (GUI), které by umožňovalo vizualizaci dat pro jednodušší orientaci ve výsledcích.

Tato data by mělo být možné také zobrazit zpětně pro případné porovnání výsledků naměřených na různých platformách.



Obr. 8.1: Diagram průběh testování nástrojem.

9 Vývojové prostředí

K vývoji nástroje (aplikace) je nutné zvážit volbu vhodného vývojového prostředí vzhledem k požadavkům, které jsou na výslednou aplikaci kladeny. Od tohoto výběru se odvíjí následné možnosti implementace aplikace a její potencionální schopnosti, případně nedostatky.

9.1 Programovací jazyk

Programovací jazyk je prostředek zápisu instrukcí určených k vykonání výpočetním systémem. Existuje několik typů programovacích jazyků, které jsou děleny na základě míry abstrakce ulehčující práci programátora při tvorbě.

První kategorií programovacích jazyků vhodných pro tvorbu testovacího nástroje jsou nižší programovací jazyky, obecně známé jako „úzce spjaté s hardwarem“. Jedná se o prostředky umožňující zadávání přesných instrukcí výpočetní jednotce pro precizní práci. Příkladem z této kategorie může být například jazyk C. Způsob tvorby programu pomocí těchto jazyků je však velmi časově náročný a vyžaduje notnou znalost programovacího jazyka a technické konstrukce cílové platformy.

Druhou kategorií jsou vyšší programovací jazyky. Do této skupiny spadá většina programovacích jazyků, převážně se však jedná o jazyky C++, C#, Java a Python. Tyto jazyky poskytují značně větší abstrakci nad jednotlivými instrukcemi zadávanými výpočetní jednotce. Další výhodou těchto jazyků je možnost vytváření objektů umožňujících ještě větší abstrakci a usnadnění tvorby programů.

Při výběru vhodného programovacího jazyka pro tuto práci byly zohledněny především zkušenosti autora s programovacím jazykem, počet knihoven vytvořených pro daný jazyk využitelných při tvorbě testovacího nástroje a způsob implementace výsledné aplikace na cílové platformy s daným operačním systémem.

Zvoleným programovacím jazykem pro vývoj navrženého nástroje se stal jazyk Python z důvodu zahrnutí tohoto jazyka jako základního prvku v distribucích operačního systému Linux, široké nabídky využitelných knihoven a větší předchozí zkušenosti autora s tímto jazykem.

9.2 Vývojářská aplikace

S vybraným jazykem pro tvorbu se pojí vývojové prostředí schopné kompilovat vytvořené seznamy instrukcí a efektivně je testovat. V případě jazyka Python jsou primární vývojářské aplikace vytvořené společnostmi Microsoft a JetBrains.

PyCharm je aplikace vytvořená pro tvorbu aplikací v jazyce Python. Jedná se o komplexní vývojářský nástroj umožňující tvorbu aplikací. Nabízí velkou škálu nástrojů na úkor velikosti a složitosti aplikace.

Microsoft Visual Code je editor zdrojového kódu, který je možné díky rozšířením upravit na editor většiny programovacích jazyků. Tuto aplikaci je možné využít také k úpravě velkého množství souborů, což může být nápomocné v případně vytvoření postupu zpracování a uchování výsledných dat. Podstatnou vlastností je také malá velikost aplikace umožňující její instalaci na embedded systémy pro případ nutnosti úpravy zdrojového kódu v rámci testování výsledné aplikace.

Z těchto důvodů byl zvolen jako vývojářské prostředí produkt MS Visual Code, který byl doplněn o rozšíření pro práci s programovacím jazykem Python.

9.3 Vývojová a testovací platforma

Samotná aplikace byla vyvíjena na operačním systému Microsoft Windows 10 Home, přičemž cílovým operačním systémem byla distribuce systému Linux s názvem Raspbian OS určená pro embedded platformy bristské společnosti Raspberry [32]. Rozdíl mezi vývojovou a testovanou platformou způsobil problémy při vývoji a následném testování aplikace.

Hlavním problémem byly rozdíly ve stavbě a fungování operačních systémů, což zvýšilo časovou náročnost testování vyvíjeného nástroje a nutnost adaptace nástroje pro správné fungování na obou platformách.

Dalším problémem způsobeným touto rozdílností platforem byla implementace knihoven do zdrojového kódu aplikace. Tyto knihovny musely být také optimalizovány k práci na obou operačních systémech, což způsobilo jistá omezení při výběru knihoven k použití (příkladem tohoto problému může být knihovna `energyusage` optimalizovaná pouze na distribuce operačního systému Linux).

Výsledné sestavení nástroje bylo nutné provést na již zmíněném operačním systému Raspbian z důvodu zamezení potencionálním problémům při využívání aplikace na této platformě.

9.3.1 Embedded platformy pro testování

Hardware určený k testování softwarového nástroje je vyráběn společností Raspberry Pi Foundation. Konkrétněji se jedná o produkty zmíněné v kapitole 5.2.

Tyto platformy byly následně osazeny paměťovými kartami s příslušnými verzemi operačních systémů Raspbian a k nim byly připojeny nutné periferie k jejímu ovládání [33]. Mezi periferie zařízení se řadí monitor, klávesnice a počítačová myš,

které jsou považovány za základní ovládací prvky tohoto systému. V případě konzolové verze operačního systému se jedná pouze o monitor a klávesnici k zadávání příkazů. Monitor byl připojen k platformě Raspberry Pi pomocí HDMI kabelu (také miniHMDI či microHDMI), přičemž k připojení ostatních periferií byly využity USB konektory poskytované platformou [34]. Napájecím prvkem celého systému se stal adaptér uzpůsoben k aktuálně používané verzi přístroje Raspberry Pi.

10 Aplikace CipherBench

Tvorba aplikace s názvem „CipherBench“ byla hlavní náplní praktické části této diplomové práce. Jedná se o automatizovaný nástroj k testování vybraných kryptografických primitiv (viz kapitola 4.1) dle aktuálních nařízení a doporučení národních a nadnárodních institucí k zajištění standardu bezpečnosti výpočetních technologií.

10.1 Implementace kryptografických primitiv

Základním prvkem vytvořené aplikace jsou implementace jednotlivých kryptografických primitiv. Jedná se o zdrojové kódy jednotlivých algoritmů v daném jazyce připravené k jejich dalšímu využití, například v obsáhlejší aplikaci. Jednou z možností jejich implementace je využití knihoven, které jsou optimalizovány pro tento způsob užití. Tyto knihovny je možné importovat do vlastního zdrojového kódu a pomocí specifických příkazů spustit metody obsažené uvnitř knihovny.

Méně efektivním způsobem implementace algoritmu je vložení celého zdrojového kódu primitiva do své aplikace. Tento způsob implementace je však neefektivní z důvodu horší optimalizace algoritmu a absence kontroly bezpečnosti a správnosti fungování zdrojového kódu. Volba vložení celého kódu je vhodná pouze v krajním případě.

10.1.1 Kryptografické knihovny

Při vývoji aplikace CipherBench byly ve velké míře využity přední kryptografické knihovny uzpůsobené pro implementaci v jazyce Python. Primárním zdrojem algoritmů se stala knihovna `PyCryptography` [35]. Jedná se o implementace algoritmů dle předem stanovených požadavků americkou institucí NIST [8], které byly následně testovány širší veřejností, a jsou považovány za ověřený zdroj pro práci s kryptografickými primitivy. Výhodou užívání této knihovny byla také velmi dobře zpracovaná dokumentace s ukázkami užívání algoritmů a popisem jejich možných variant.

Pomocí knihovny PyCryptography byly implementovány následující algoritmy.

- Symetrické algoritmy:
  - AES (verze v módu CBC s klíči o velikosti 128b, 192b a 256b),
  - 3DES (verze v módu CBC s klíči o velikosti 128b, 192b a 256b),
  - Camellia (verze v módu CBC s klíči o velikosti 128b, 192b a 256b),
  - ChaCha20 (s klíčem o velikosti 265b).
- Módy šifer (velikost klíče je 256b):
  - Šifra AES v módech:
    - mód ECB,

    - mód CBC,
    - mód CFB,
    - mód OFB,
    - mód CTR,
    - mód GCM,
    - mód CCM,
  - ChaCha20 v módu Poly1305.
- Algoritmy autentizačních technik:
  - HMAC (s velikostí klíče 64b),
  - HMAC-SHA1 (s velikostí klíče 112b),
  - CMAC-AES (s velikostí klíče 128b),
  - CMAC-TDES (s velikostí klíče 128b).
- Hashovací algoritmy:
  - SHA1,
  - SHA2-256,
  - SHA2-384,
  - SHA2-512,
  - SHA3-256,
  - SHA3-384,
  - SHA3-512.
- Algoritmy digitálního podpisu:
  - DSA (s velikostí klíče 2048b a 3072b),
  - ECDSA (s křivkou SECP-256R1),
  - RSA (s velikostí klíče 2048b a 3072b).
- Algoritmy distribuce klíčů:
  - DH (s velikostí klíče 2048b a 3072b),
  - ECDH (s křivkami SECP-224R1 a SECP-256R1),
  - RSA KEM (s velikostí klíče 2048b a 3072b).
- Generátory náhodných čísel:
  - Hash DRBG,
  - CTR DRBG (se šifrou AES-128),
  - CTR DRBG (se šifrou TDES-128).

10.1.2 Implementace vlastních algoritmů

Jak již bylo zmíněno v podkapitole 10.1, jedná se o způsob implementace, který by neměl být aplikován k využití v reálných komunikačních systémech. V případě některých algoritmů je však tato volba nevyhnutelná z určitých důvodů.

Pro uchýlení se k tomuto rozhodnutí předcházel časově náročný průzkum zdrojů

s cílem nalezení předem vyhotovené a ověřené implementace algoritmu. Tento výzkum však byl v malém množství případů neúspěšný. Jedná se o algoritmy:

- Twofish (s velikostí klíče 128b, 192b a 256b) [36],
- EC Schnorr (s křivkou SECP-256R1) [37],
- HMAC DRBG [38].

Algoritmy implementované tímto způsobem byly vybrány z veřejného internetového úložiště GitHub [39]. Z tohoto důvodu se však nejedná o algoritmy vhodné pro využití v reálných systémech. V případě této práce se jedná pouze o test provedení algoritmů.

10.1.3 Chybějící implementace primitiv

V malém množství případů však také nastala situace, kdy nebyla k nalezení volně dostupná implementace algoritmu v podobě zvoleného programovacího jazyka Python. Jedná se o algoritmy:

- Serpent,
- Snow 2.0 a Snow 3G,
- PSEC-KEM a ACE-KEM.

Důvody k vynechání výše zmíněných algoritmů jsou vysvětleny v následující části této podkapitoly. V případě pokračování ve vývoji testovacího nástroje CipherBench by implementace těchto kryptografických primitiv byla velkým přínosem.

Algoritmus Serpent

K nalezení byly dvě vlastní implementace tohoto algoritmu v jazyce Python, avšak obě verze byly velmi zastaralé a bohužel také nefunkční či nedokončené (verze Pythonu 1.5 a 2.5) [40]. Po dalším zkoumání bylo toto primitivum nalezeno v knihovnách jiných jazyků, převážně v jazyce C.

Pro použití metod z jiných programovacích jazyků existuje knihovna jménem `ctypes` umožňující toto volání uskutečnit pomocí vytvoření dynamické knihovny na cílovém operačním systému [41]. Tento postup však nebyl úspěšný z důvodu složitosti vstupních parametrů pro volání metod skrze knihovnu `ctypes`, která tyto vstupní parametry bohužel nepodporuje. Dalším problémem při využití tohoto řešení je nutnost omezení vytvořeného nástroje pouze na jeden operační systém (dynamické knihovny využívané touto knihovnou v OS Windows jsou jinak stavěné a strukturované než v OS Linux).

Po tomto časově náročném zkoušení možností zprovoznění algoritmu Serpent bylo v rámci dokončení práce od tohoto kryptografického primitiva upuštěno. Podporujícím důvodem k tomuto rozhodnutí byla nízká četnost používání tohoto algoritmu v praxi.

Snow 2.0 a Snow 3G

Algoritmy Snow 2.0 a Snow 3G jsou proudové šifry využívané především v telekomunikačních technologiích. Z důvodu tohoto specifického využití kryptografického primitiva nebyly nalezeny žádné veřejné zdroje implementace, a tudíž byly tyto algoritmy z praktické části taktéž vypuštěny.

ACE-KEM a PSEC-KEM

Zkratka KEM (angl. Key Encapsulation Mechanism) označuje kryptografická primitiva určená k zabezpečení klíče symetrických šifer pro přenos pomocí asymetrické kryptografie založené na eliptických křivkách. Jedná se o varianty algoritmů vycházejících z algoritmu ECDH. Jde však o velmi specifické algoritmy jejichž implementací není mnoho. Ze zmíněných důvodu byl implementován primárně algoritmus ECIES-KEM, který je považován za nejznámější algoritmus ze skupiny těchto specifických mechanismů.

10.1.4 Obecná struktura algoritmu ve zdrojovém kódu

Zdrojové kódy algoritmů se ve vytvořené aplikaci liší vstupními parametry a volanými metodami. K docílení větší přehlednosti kódu byly jednotlivé implementace algoritmů psány s cílem vytvoření co nejpodobnější struktury. Výsledkem této snahy je možnost představení obecné struktury metody uvnitř aplikace, která se liší pouze v závislosti na rozdílu způsobu implementace algoritmu samotného.

Výpis 10.1: Implementace algoritmu AES.

```
1  def AES(self, n, warm_up = False):
2      try:
3          #import libraries
4          import os
5          from cryptography.hazmat.primitives.ciphers import
6              Cipher, algorithms, modes
7          from utils import CheckIt, StartCPUTime,
8              StopCPUTime, GetCPUTime, CalculateSpeed,
9              CalculateMemory, PrintResults
10
11         #save start time of enc
12         start_time = StartCPUTime()
13         #generate params
14         key = os.urandom(n)
15         iv = os.urandom(16)
16         #perform encryption
17         cipher = Cipher(algorithms.AES(key), modes.CBC(iv))
18         encrypt = cipher.encryptor()
```

```
19          ct = encrypt.update(lorem_ipsum)+encrypt.finalize()
20          #save stop time of enc
21          time_enc = StopCPUTime(start_time)
22
23          #save start time dec
24          start_time = StartCPUTime()
25          #perform decryption
26          decrypt = cipher.decrypt()
27          output = decrypt.update(ct)+decrypt.finalize()
28          #save stop time of dec
29          time_dec = StopCPUTime(start_time)
30
31          #assign all neccessary params for print/save file
32          name = 'AES'
33          key_size = str(n*8)
34          #call method for check of the result
35          result = CheckIt(lorem_ipsum, output)
36          #call methods for computing time
37          cpu_enc, part1 = GetCPUTime(time_enc, 'encryption')
38          cpu_dec, part2 = GetCPUTime(time_dec, 'decryption')
39          #call methods for calculation of speed
40          speed_enc = CalculateSpeed(lorem_ipsum, time_enc)
41          speed_dec = CalculateSpeed(ct, time_dec)
42          #call method for calculation of memory
43          mem = CalculateMemory()
44      except:
45          #if any of method used above throws an error,
46          #params for print/save will attain values bellow
47          name = 'AES'
48          key_size = str(n*8)
49          result = 'FAIL'
50          part1 = 'encryption'
51          part2 = 'decryption'
52          cpu_enc, cpu_dec, speed_enc, speed_dec, mem = '-',
53              '-', '-', '-', '-'
54      finally:
55          #if fork for use cases when application wants to
56          #test run the algorithm
57          if(warm_up == True):
58              return None
59          else:
60              #calling the method for generating console
61              #output
62              PrintResults(self, name, key_size, result,
63                  cpu_enc, part1, cpu_dec, part2, speed_enc,
64                  speed_dec, mem)
65              #returning measured values for later
```

```
66              #storing in separate file
67              return name, key_size, result, cpu_enc,
68                  cpu_dec, speed_enc, speed_dec, mem
```

Výpis algoritmu (znázorněn ve Výpisu 10.1) začíná zadefinováním funkce s požadovanými parametry (řádek 1). Následuje obalení hlavní části kódu do bloků *try* a *except*. Jedná se o bezpečnostní prvek umožňující testování bloku kódu, zda neobsahuje chyby zamezující správnému průběhu metody. V případě nalezení chyby je místo normálního průchodu metodou vykonán kód v části *except*. Poslední částí tohoto bezpečnostního prvku je blok *finally*, jehož obsah se vykonává vždy.

Počátečními příkazy v hlavní části kódu jsou vždy importy knihoven vyžadovaných algoritmem (řádky 4 až 9). Zároveň s nimi jsou vždy importovány metody souboru *utils* obsahující funkce testovacích nástrojů pro výpis výsledků, kontrolu výsledku (viz Výpis 10.2), měření času (viz Výpis 10.3), rychlosti (viz Výpis 10.4) a využití paměti (viz Výpis 10.5). Před započetím jakýchkoliv výpočtů v rámci algoritmu je zaznamenán aktuální čas pomocí metody `StartCPUTime()` na řádku 12. Dále už následují volání metod implementovaných knihovnami sloužících k provedení operací kryptografického primitiva na řádcích 14 až 27 včetně generování náhodných počátečních vstupů algoritmu na řádcích 14 a 15. Závěrečnou částí metody jsou výpočty a zapsání výsledků do proměnných, které jsou vráceny do hlavní metody aplikace k dalšímu zpracování (metoda *main()* v souboru *main.py*).

Výpis 10.2: Implementace metody kontroly výstupu.

```
1  #method comparing input/outpus of methods
2  def CheckIt(input = None, output = None):
3      if((input != None) and (output != None)):
4          if(input == output):
5              return 'SUCCESS'
6          else:
7              return 'FAIL'
8      if((input == None) and (output == None)):
9          return 'SUCCESS'
10     else:
11         return 'FAIL'
```

Výpis 10.3: Implementace metod měření času.

```
1  #method returns time at the moment
2  def StartCPUTime():
3      start = perf_counter_ns()
4      return start
5
```

```
#method for computing difference in time (ns)
def StopCPUTime(start):
    stop = perf_counter_ns()
    delta = (stop - start)
    return delta

#method returning correct format of CPU time (s)
def GetCPUTime(delta, part=None):
    message = str(delta / 1000000000) +'s'
    return message, part
```

Výpis 10.4: Implementace metody výpočtu rychlosti.

```
#method for calculation of speed (kB/s)
def CalculateSpeed(data, time):
    index = 1 / (time / 1000000000)
    speed = round((len(data) * index)/1000, 4)
    return str(speed) + "kB/s"
```

Výpis 10.5: Implementace metody měření paměti.

```
#method for calculation of memory (B)
def CalculateMemory():
    mem = tracemalloc.get_traced_memory()
    tmp_mem = mem[1]
    ResetTrace()
    return str(tmp_mem) + "B"
```

10.2 Nástroje pro měření algoritmů

V sekci 8.1 byly zmíněny hlavní měření, které by výsledná aplikace měla zaznamenávat. Naměřená data by měla obsahovat údaje o času nutném pro úspěšné provedení algoritmu, rychlosti provedení algoritmu v poměru k časové jednotce, paměti využité systémem při průběhu algoritmu a energii spotřebované platformou při provádění testu. Následující podkapitoly se zabývají způsoby implementace těchto měřících nástrojů.

10.2.1 Měření času

Časová náročnost je v dnešní době důležitým faktorem při výběru vhodné výpočetní platformy v kombinaci s vhodnou implementací požadované aplikace. Tento časový

údaj je možné měřit jako celkový čas nutný k dokončení instrukcí systémem (angl. Wall time), nebo čas strávený výpočetní jednotkou systému na daném úkonu (angl. CPU time).

V případě vývoje testovacího nástroje CipherBench se jedná o měření doby strávené vykonáním kryptografického primitiva procesorem. Toto měření je uskutečněno pomocí knihovny `time`, jež je základní součástí aplikace Python [42].

Konkrétněji se jedná o využití metody `perf_counter_ns`, která vrací aktuální čas v jednotkách nanosekund. Celý proces měření času spočívá v zaznamenání počátečního času před začátkem generování potřebných parametrů daného algoritmu a zaznamenání konečného času po vytvoření výstupu algoritmu. Časový údaj je poté tvořen rozdílem konečného a počátečního záznamu, přičemž je tento údaj následně převeden do sekund se zaokrouhlením na sedm desetinných míst pro lepší prezentaci výsledku pro koncového uživatele (viz Výpis 10.3).

Algoritmy dělící se na části tvorby a ověření (např. šifrování a dešifrování, či podepsání a ověření) obsahují tyto měření dvě - pro každou část zvlášť. Tyto údaje jsou také separátně zaznamenávány a ukládány.

10.2.2 Měření rychlosti

Záznamy o rychlosti daného algoritmu jsou vypočítávány po jeho dokončení pomocí vyhodnocení potenciálního počtu provedení algoritmů za jednu sekundu. Vstupní hodnota času této operace je v jednotkách nanosekund pro větší přesnost výsledku.

$$index = \frac{1}{\frac{time}{10^9}} \tag{10.1}$$

Tato hodnota je dále vynásobená délkou vstupních dat, které má algoritmus za cíl zpracovat, a převedena na jednotku kB/s se zaokrouhlením na čtyři desetinná místa.

$$rychlost = index * data \tag{10.2}$$

Hodnoty u algoritmů skládající se z více částí jsou počítány a ukládány taktéž zvlášť

10.2.3 Měření paměti

Měření paměti se může stát komplikovanou záležitostí z pohledu své definice v rámci požadovaných výsledků a vlastní implementace. Za sledované měření v případě tohoto testovacího nástroje byla zvolena paměť využívaná algoritmem k ukládání proměnných pro úspěšné provádění svých předdefinovaných instrukcí.

Prvotní přístup k implementaci tohoto nástroje měření bylo využití knihovny `memory_profiler` [43]. Jedná se o knihovnu umožňující detailní popis paměťové náročnosti jednotlivých částí zdrojového kódu, kterou je možné do daného kódu

programu či aplikace také zakomponovat. K implementaci je však v rámci aplikace nutné využít dekorativních funkcí, které Python nabízí.

Dekorativní funkce jsou speciálním typem metod umožňující zaobalení volání jiné metody do volání vlastního, přičemž se nejdříve vykoná zavolání dekorativní funkce a až poté probíhá volání metody původní. Spuštěná metoda je v tomto případě volána jako potomek dekorativní funkce, a tedy umožňuje dekorativní metodě přístup k většímu množství údajů.

Problémem tohoto přístupu však bylo odesílání naměřených dat zpátky do základní části aplikace k dalšímu zpracování. Jelikož se jedná o speciální typ metody, přicházejí s jejími novými možnostmi také nová omezení. Dále využití této funkce znamenalo zaobalení každé funkce, a tedy zvýšení komplikovanosti výsledné aplikace.

Finální verze aplikace využívá knihovnu `tracemalloc` [44]. Umožňuje jednodušší práci programátora při vytváření výpisu a porovnání spotřeby paměti v různých částech zdrojového kódu při průběhu aplikace. Způsob fungování této knihovny se skládá z malého množství příkazů. Dvěma hlavními příkazy jsou zapnutí a vypnutí sledování paměti touto knihovnou (funkce `start()` a `stop()`). Dalším hojně používaným příkazem je `get_traced_memory()`, který po svém zavolání vrací údaje o aktuální vytíženosti paměti a maximální vytíženosti paměti zaznamenanou od posledního volání knihovny, čehož je užíváno právě v aplikaci CipherBench.

Původní myšlenkou implementace této knihovny bylo zaznamenání rozdílu maximální a aktuální hodnoty, která však byla neúspěšná z důvodu způsobu promazávání dat ze strany programu Python spouštějící vytvořenou aplikaci (tzv. Garbage collector). Z těchto důvodů je ve finální aplikaci zobrazována jako výsledek maximální hodnota spotřeby paměti zaznamenána po dobu průběhu algoritmu.

10.2.4 Měření spotřeby

Na začátek této sekce je nutné poznamenat, že se nejedná o měření implementované ve stávající verzi aplikace CipherBench. Při vývoji aplikace byly vybrány vhodné knihovny pro implementaci se jmény `pyRAPL` [45] a `energyusage` [46], které se ukázaly jako nevhodné či nefunkční.

Knihovna `pyRAPL` disponuje otevřeným zdrojovým kódem na veřejném úložišti GitHub. Nevýhodou tohoto způsobu řešení je opět nutnost využití dekorativních funkcí obdobně jako v případě zmíněném v podkapitole 10.2.3.

Druhým způsobem řešení bylo využití knihovny `energyusage`. Tato knihovna slibuje jednoduchá měření využívající jako vstupní parametr celou funkci, kterou následně svými metodami otestuje. Problémy nastaly při limitaci správného fungování pouze na operačních systémech využívajících Linuxové jádro (tzv. kernel).

Zprovoznění však nebylo možné ani na linuxovém operačním systému v případě spuštění samotného příkladu uvedeného na stránkách dokumentace knihovny.

10.3 Uchování výsledků

Aplikace CipherBench byla po konzultaci s vedoucím práce vyvíjena způsobem umožňujícím ukládání a vyvolání výsledků měření vytvořených aplikací. Jak již bylo zmíněno v sekci pojednávající o teorii výsledného nástroje (viz podkapitola 8.1), existuje více způsobů a souborů vhodných pro implementaci této funkcionality.

V případě zmíněné aplikace byla implementována funkcionalita ukládání skrze nativní knihovnu `xml.dom` a její objekt nazvaný `minidom` [47]. Ve spolupráci s těmito nástroji byly v aplikaci vytvořeny metody umožňující ukládání a načítání výsledných dat do souboru XML, jenž je možné později načíst a zobrazit. Tento soubor je vytvořen na konci testování, kdy jsou veškerá zaznamenaná data přeložena do formy vhodné pro uchování v souboru.

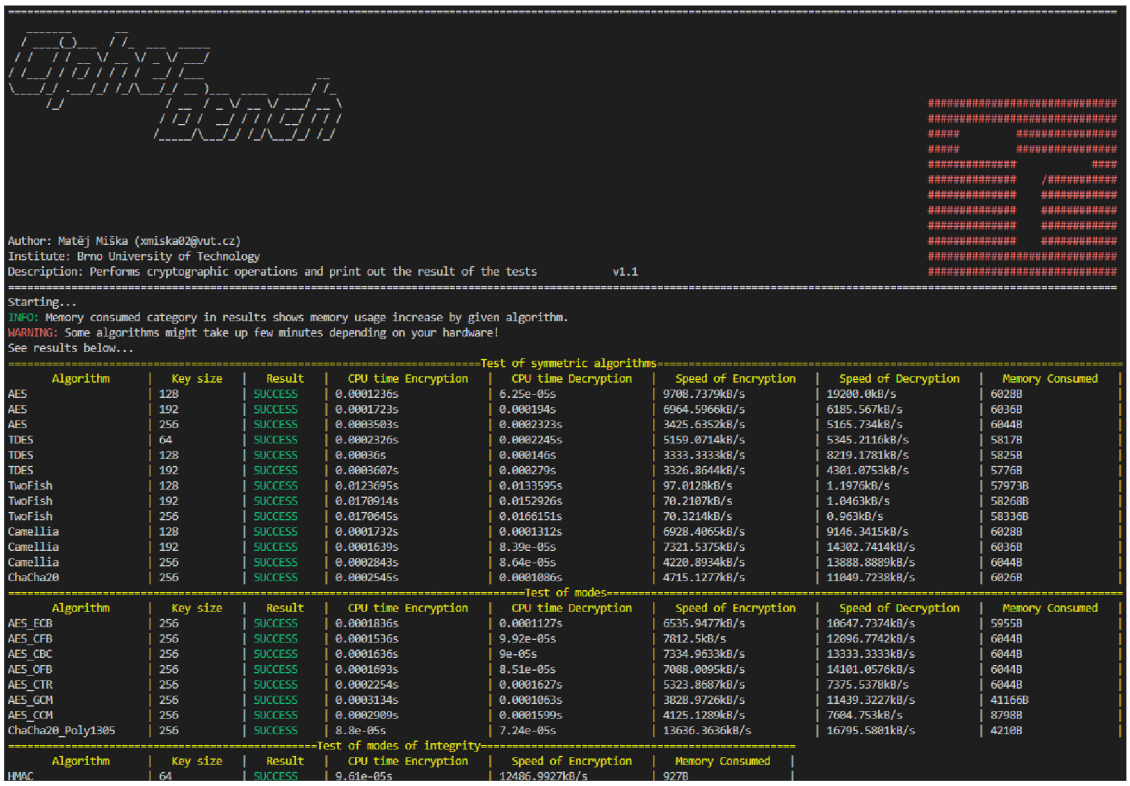
10.4 Prezentace výsledků

V návaznosti na předchozí sekci věnující se způsobu uchovávání výsledků (viz podkapitola 10.3) se tato část práce věnuje prezentaci dosažených výsledků. Požadavkem na aplikaci byla schopnost data určitým způsobem vykreslit pro jejich lepší vizualizaci a jednodušší porovnání testovaných platforem.

10.4.1 Konzolový výpis

Nejjednodušším typem aplikace je aplikace konzolová. Jedná se o odlehčenou verzi většího programu určenou k využití na systémech s nižší výpočetní silou a softwarovým vybavením, umožňující spuštění aplikace ze systémové konzole. V případě spuštění této verze aplikace jsou výsledky testování algoritmů vypisovány v reálném čase. Po dokončení testu jsou všechna data opět zapsána do souboru, který umožňuje vykreslení výsledků v aplikaci.

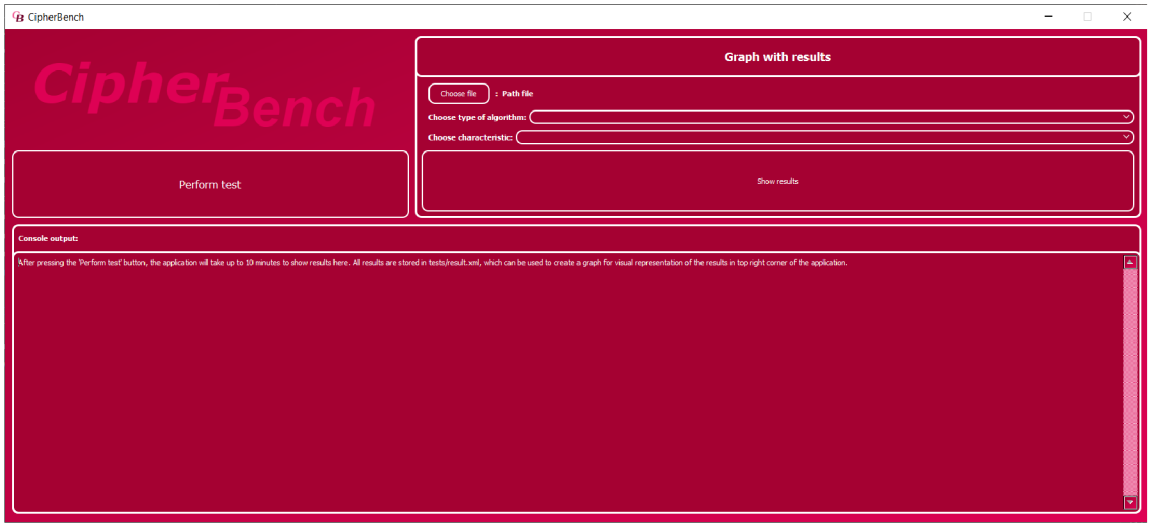
Zároveň byly vytvořeny dvě verze této aplikace. Rozdílem je barevný (viz Obr. 10.1) nebo černobílý výpis do konzole, který je uzpůsobený pro rozdílné operační systémy. V případě OS Windows je doporučené využívat černobílou verzi z důvodu nutnosti úpravy konzole pro užití nadefinovaných barev, zatímco konzole OS Linux podporuje barevný výpis nativně.



Obr. 10.1: Ukázka konzolového výpisu aplikace CipherBench.

10.4.2 Grafické rozhraní

Rozšířená verze algoritmu CipherBench je obohacená o grafické rozhraní umožňující vykreslení výsledků v grafické podobě dle Obr. 10.2.

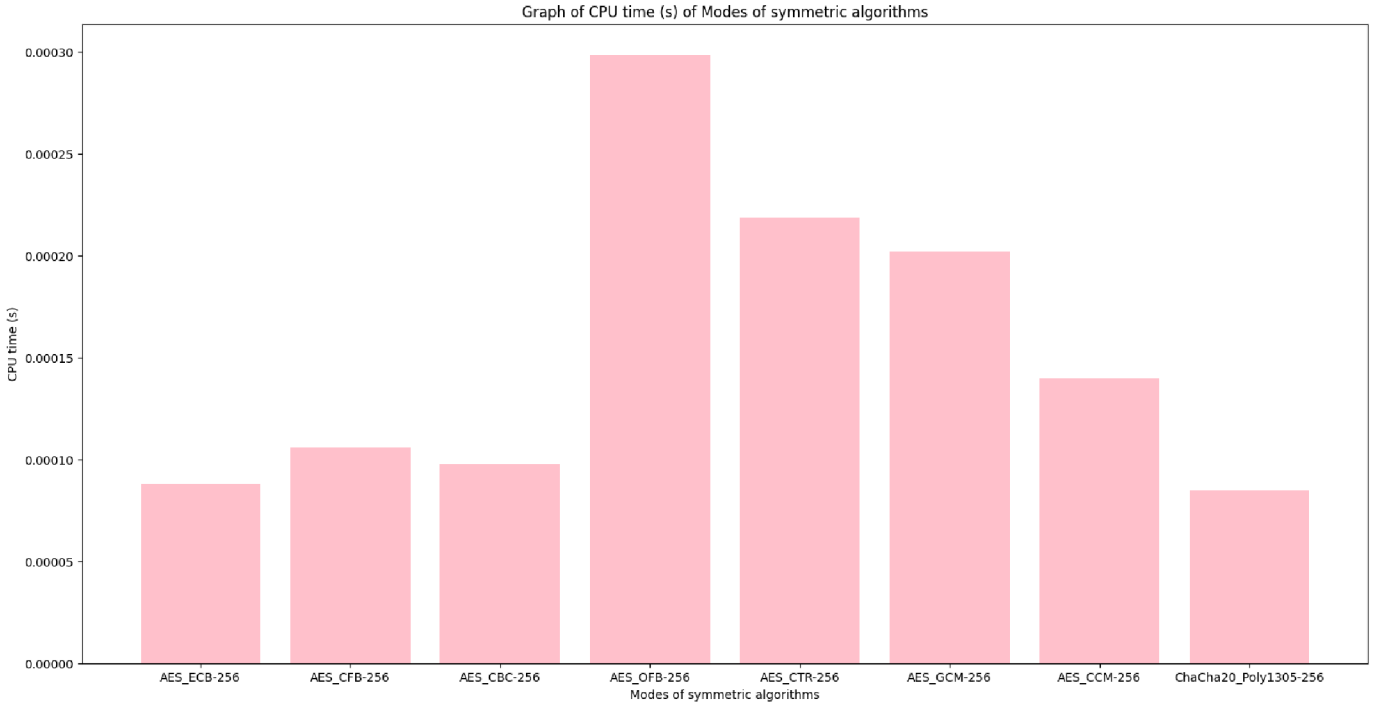


Obr. 10.2: Uživatelské rozhraní aplikace CipherBench.

Samotný návrh rozhraní byl vytvořen pomocí aplikace Qt Designer [48] poskytu-

jící vývojové prostředí k vytváření a stylizování GUI pro aplikace v jazyce Python. Tato aplikace generuje soubor s příponou UI obsahující informace o přiložených souborech (například soubory PNG) využitých v designu aplikace. Následně byl tento hlavní soubor přeložen do jazyku Python pomocí knihovny `PyQt5` [49], který byl poté vložen do složky projektu a provázán se spouštěnou částí zdrojového kódu aplikace.

Vykreslení je implementováno skrze načtení naměřených dat zpět do aplikace a vytvoření sloupcového grafu dle požadovaných dat měření a typu algoritmů (viz Obr. 10.3). Tento graf může být dále zkoumán či uložen do podporovaného formátu (například formáty PNG, JPG, PDF či SVG). Grafické rozhraní poskytuje také zkrácený výpis konzole pro okamžitou kontrolu, zda testování daného algoritmu proběhlo v pořádku a jaké hodnoty byly naměřeny.



Obr. 10.3: Grafické znázornění výsledků aplikací CipherBench.

11 Užití aplikace

Spuštění konzolové verze aplikace je prováděno pomocí příkazu znázorněného ve Výpisu 11.1. Tento příkaz spouští hlavní soubor aplikace CipherBench. Různé možnosti volání programu z konzolové řádky jsou vypsány při zadání příkazu s argumentem `-h` nebo `--help`. Při spuštění aplikace s těmito argumenty jsou uživateli následně vypsány do konzole všechny možnosti zadání argumentů aplikaci. Jedná se o argumenty:

- `-T` nebo `--test`, která spustí hlavní test aplikace (viz Výpis 11.1),
- `-G` nebo `--graph`, která při zadání typu algoritmů, požadovaných dat a souboru s daty vykreslí graf výsledků (viz Výpis 11.2),
- `-S` nebo `--save`, která při zadání typu algoritmů, požadovaných dat, souboru s daty a požadovaného názvu nového souboru uloží data zobrazená v grafu do souboru zvoleného formátu (viz Výpis 11.3).

Výpis 11.1: Příkaz spuštění testu aplikace CipherBench.

```
$ python3 main.py -T
```

Ke spuštění aplikace s argumenty pro vykreslení grafu či jeho uložení je nutné jejich správné zadání. Správné pořadí a formát zadávání jsou znázorněny ve Výpisu 11.2 pro vytvoření grafu aplikací a ve Výpisu 11.3 pro vytvoření a uložení grafu do souboru PNG.

Výpis 11.2: Správné zadání argumentů pro zobrazení grafu.

```
$ python3 main.py -G typ_algoritmu data cesta_souboru_s_daty
```

Výpis 11.3: Správné zadání argumentů pro uložení grafu.

```
$ python3 main.py -S typ_algoritmu data cesta_souboru_s_daty
nazev_noveho_souboru
```

Možnosti zadání typu argumentu aplikace jsou:

- Symmetric_algorithms - pro symetrické algoritmy,
- Modes_of_symmetric_algorithms - pro módy symetrických algoritmů,
- Modes_of_integrity - pro algoritmy autentizačních technik,
- Hash_algorithms - pro hashovací algoritmy,
- Signature_algorithms - pro algoritmy digitálních podpisů,
- Key_exchange_algorithms - pro algoritmy výměny klíčů,
- Random_number_generators - pro generátory náhodných čísel.

V rámci zadávání požadovaných dat měření se jedná o argumenty:

- CPU_enc - pro čas využitý k šifrování, hashování, podepisování či generování algoritmu,

- CPU_dec - pro čas využitý pro sekundární část, pokud ní algoritmus disponuje (dešifrování a ověřování),
- speed_enc - pro rychlost šifrování, hashování, podepisování či generování algoritmu,
- speed_dec - pro rychlost sekundární části, pokud ní algoritmus disponuje (dešifrování a ověřování),
- memory - pro maximální spotřebu paměti využité po dobu průběhu algoritmu.

V případě zadání příkazu s argumentem, jež daný algoritmus nemá, je automaticky přeložen na argument podporovaný daným algoritmem (například překlad argumentu CPU_dec na CPU_enc u hashovacích funkcí).

Pro využití těchto funkcí ve verzi aplikace s grafickým rozhraním je vytvořeno intuitivní uživatelské rozhraní (viz podkapitola 10.4.2). Ke zprovoznění této verze aplikace je však třeba doinstalovat knihovnu `PyQt5` umožňující její zobrazení. Knihovna `PyQt5` měla být původně součástí složky aplikace jako ostatní externí knihovny, avšak pro její správné fungování je třeba vždy knihovnu sestavit před instalací. Tato instalace je provedena pomocí příkazu znázorněného ve Výpisu 11.4. Uživatelské rozhraní aplikace se po úspěšné instalaci knihovny spouští pomocí souboru *CipherBench.py* (viz Výpis 11.5). Grafické rozhraní je uzpůsobené spíše pro využití na výkonnějších systémech a slouží k jednodušší orientaci uživatele neznalého aplikace k jejímu testování. Z těchto důvodů je instalace knihovny pro grafické rozhraní ponechána na potřebách uživatele aplikace.

Výpis 11.4: Instalace knihovny grafického rozhraní aplikace.

```
1    $ sudo apt-get install python3-pyqt5
```

Výpis 11.5: Spuštění uživatelského rozhraní aplikace.

```
1    $ python3 CipherBench.py
```

11.1 Souborová struktura aplikace

Veškeré soubory aplikace jsou uchovávány ve složce pojmenované *verze*, přičemž názvy složek uvnitř jsou upraveny dle verze aplikace pro jednodušší orientaci uživatele. V jednotlivých podsložkách se nachází také hlavní soubor *main.py* (případně *CipherBench.py* či *main.exe*) určený ke spuštění aplikace.

Uvnitř dílčích složek se nachází soubory v jazyce Python obsahující implementace algoritmů roztříděné dle jejich typu. V případě, že je soubor nazván jménem algoritmu, jedná se o vlastní implementaci algoritmu, která je využívána při testování.

Dále kromě složek vytvořených samotnou aplikací při jejím sestavení je součástí také složka *tests*, do které jsou ukládány výsledky testů. V případě opakovaného testování aplikací bude původní uložený soubor přepsán novými výsledky. K finálnímu uložení výsledků řešení je doporučené daný soubor přejmenovat či uložit na jiné místo.

Poslední součástí přílohy této práce jsou externí knihovny nutné pro správný průběh aplikace bez nutnosti instalace knihoven na testovaný systém (složka *lib*). Tyto knihovny byly vytvořeny na přístroji Raspberry Pi 4 (viz Obr. 5.5) a v případě využití aplikace na odlišné platformě je doporučeno dané knihovny doinstalovat (podle příkladu instalace ve Výpisu 11.6) na cílovém přístroji. Jedná se především o knihovny `matplotlib` [50], `ECPy` [51] a `eciespy` [52]. Pro správné fungování aplikace na platformě bez instalace těchto knihoven je nutné sestavené knihovny vložit do složky požadované verze aplikace. Potřebné knihovny tedy musí být předem vytvořené na totožné platformě pro jejich užití v novém systému. Příklad potřebných souborů knihoven k nakopírování a přiložení ke kódu je uložen ve složce *lib*.

Výpis 11.6: Příklad instalace knihovny pro vykreslení grafu.

```
1    $ pip install matplotlib
```

12 Analýza výsledků

Embedded platformy byly podrobeny testování pomocí vytvořeného nástroje CipherBench. Na vybrané platformy byl nainstalován výše zmíněný operační systém Raspbian (verze x64, případně x32), na kterém byly podniknuty veškeré nutné kroky ke spuštění nástroje. Soubory s výsledky získanými při testování se nacházejí v příloze této práce ve složce s názvem *Výsledky*.

12.1 Výsledky testování

Výsledky testování byly zaznamenány do souborů *results.xml*, jejichž název byl doplněn o název testované platformy. Z důvodu velkého množství dat je předmětem porovnávání v této kapitole pouze skupina hashovacích algoritmů, na kterých je patrný rozdíl ve výkonnosti jednotlivých platforem bez většího rozdílu dosažených výsledků zkreslujících jejich vizuální prezentaci. Pro zobrazení naměřených výsledků je možné využít aplikace CipherBench, případně vyhledat požadované údaje v souborech s těmito daty doloženými v příloze.

Paměťová náročnost není uvedená mezi následujícími výsledky, jelikož se tento údaj neměnil vzhledem k verzím hashovacího algoritmu na dané platformě. Výsledek této kategorie je zaznamenán v následující tabulce 12.1.

Tab. 12.1: Paměťová náročnost hashovacích algoritmů.

Platforma	Pi 1	Pi 2	Pi 3	Pi 4	PC
Paměťová náročnost [B]	354	354	688	688	688

Druhým měřeným údajem ve vytvořeném testovacím nástroji je časová náročnost provedení algoritmu na dané platformě zobrazená v grafech na Obr. 12.1, Obr. 12.2, Obr. 12.3, Obr. 12.4 a Obr. 12.5. Pokud se jedná o algoritmy mající dvě samostatné části nutné k jejich fungování (například šifrování a dešifrování či podepsání a ověření), je možné tyto údaje zobrazit ve dvou oddělených grafech. Na posledním grafu (12.5) je vertikální osa uváděna v sekundách s mocnitelem -5 z důvodu přesnosti takto malého časového údaje.

Posledním měřeným údajem byl přepočet rychlosti algoritmu v převodu vstupu na výstup na dané platformě (viz podkapitola 10.2.2), jež lze vidět na Obr. 12.6, Obr. 12.7, Obr. 12.8, Obr. 12.9 a Obr. 12.10. Obdobně jako u předchozího měření v případě dvou částí algoritmu je tento údaj počítán odděleně pro každou část. Horizontální osa grafu znázorňuj rychlost algoritmu v jednotkách kB/s.

12.2 Diskuse výsledků

Testování pomocí aplikace CipherBench bylo úspěšné na všech testovaných platformách. Délka tohoto testu probíhala v rámci minut u přístrojů Raspberry Pi 4 a Raspberry Pi 3, avšak v případě starších platforem (Raspberry Pi 2 a Raspberry Pi B) dosahovala celková doba testu až 4 hodin. Tato doba se odvíjela převážně od doby provedení algoritmu Diffie-Hellman, přičemž přístroji Raspberry Pi 2 trvalo provedení úkonů tohoto algoritmu 9973 sekund (přes 166 minut) při velikosti klíče 3072 bitů.

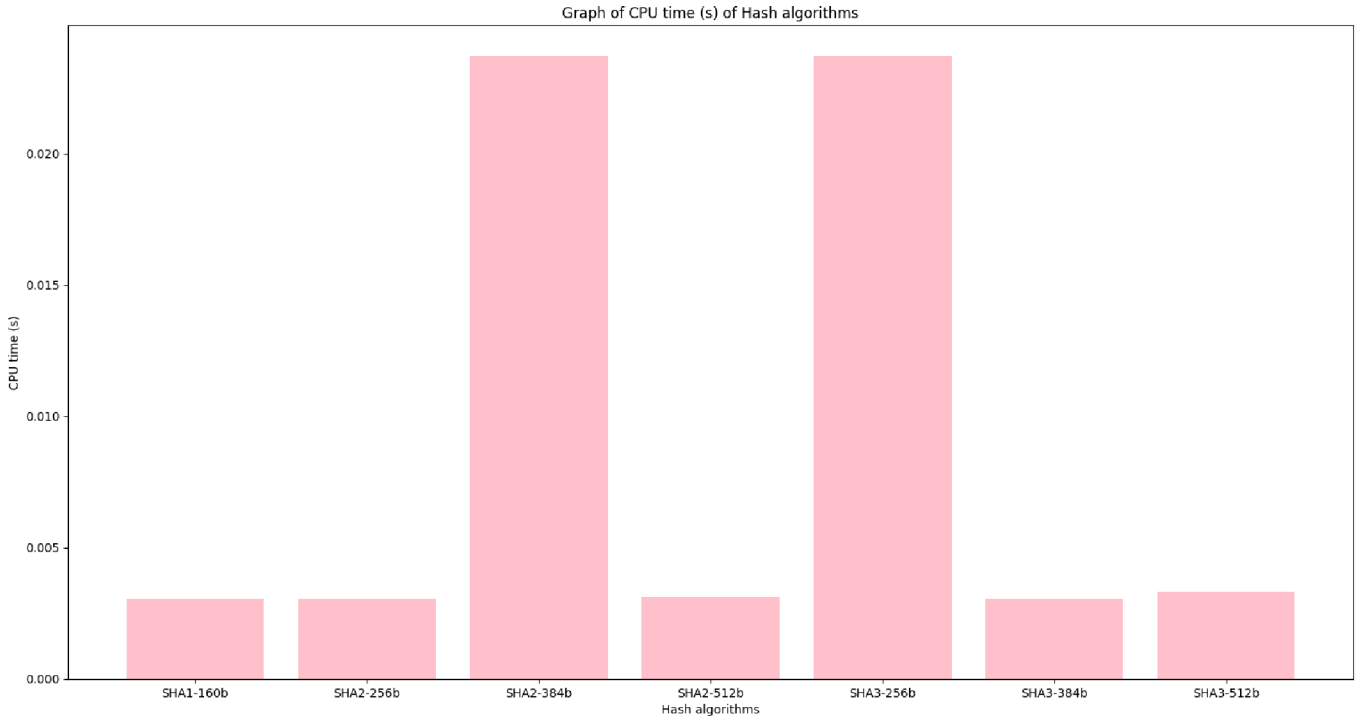
Algoritmy asymetrické kryptografie vyžadovaly na všech platformách nejvíce času. Na tomto údaji byl patrný rozdíl výkonnosti jednotlivých platforem v závislosti na době jejich výroby. Pro porovnání s výkonnějšími platformami byl spuštěn tento test také na stolním počítači s operačním systémem Ubuntu, při kterém byly naměřeny lepší výsledky v řádech jednotek až desítek sekund v porovnání s embedded zařízeními.

Jak již bylo zmíněno výše, celková doba trvání testu závisela na době provedení algoritmu Diffie-Hellman. Při zkoumání potencionálního problému zapřičiňujícím dlouhou dobu provádění algoritmu byla zjištěna nadměrná náročnost jedné konkrétní metody, a to metody *generate_parameters()*. Tato metoda zajišťuje generování potřebných dat pro následný výpočet algoritmu mající na vstupu parametry generátoru multiplikativní grupy a požadované velikosti klíče. I přes testování různých vstupů této metody nebylo možné snížit časovou náročnost této metody při požadované délce klíčů.

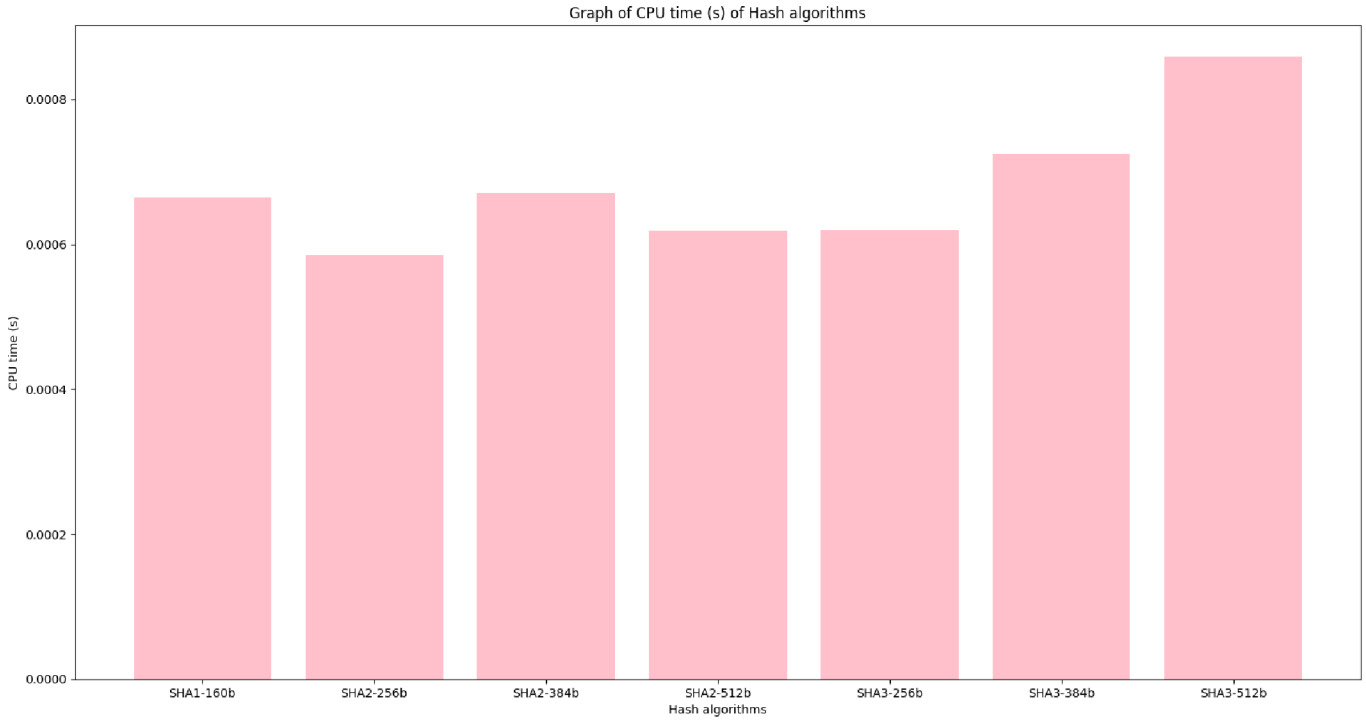
Z pohledu maximálního využití paměti (viz podkapitola 10.2.3) se jednalo o konstantní hodnotu v rámci stolního PC a přístrojů Raspberry Pi 3 a 4. V porovnání s přístroji Raspberry Pi B a Pi 2 se jednalo o menší využití paměti v řádech desítek bajtů, které v případě některých algoritmů dosahovalo až na polovinu hodnoty užívání paměti v porovnání s modernějšími modely. Tento jev je pravděpodobně způsoben rozdílným hardwarovým vybavením platforem odpovídajícím době výroby. Dalším jevem výrazně viditelným na výsledcích využívání paměti algoritmem jsou algoritmy implementované skrze vlastní zdrojový kód (viz podkapitola 10.1.2). Algoritmy takto implementované byly až desetinásobně náročnější na paměť než algoritmy implementované knihovnami.

Rychlost provádění instrukcí jednotlivých algoritmů (viz podkapitola 10.2.2) vykazovala dvojnásobné až trojnásobné zrychlení vůči přístroji předchozí generace. Toto zrychlení je odlišné v případě Raspberry Pi B, kdy se jedná o zrychlení až desetinásobné na platformě Raspberry Pi 2. V porovnání s výsledky dosaženými na stolním počítači se opět jedná až o desetinásobné zrychlení vůči platformě Raspberry Pi 4.

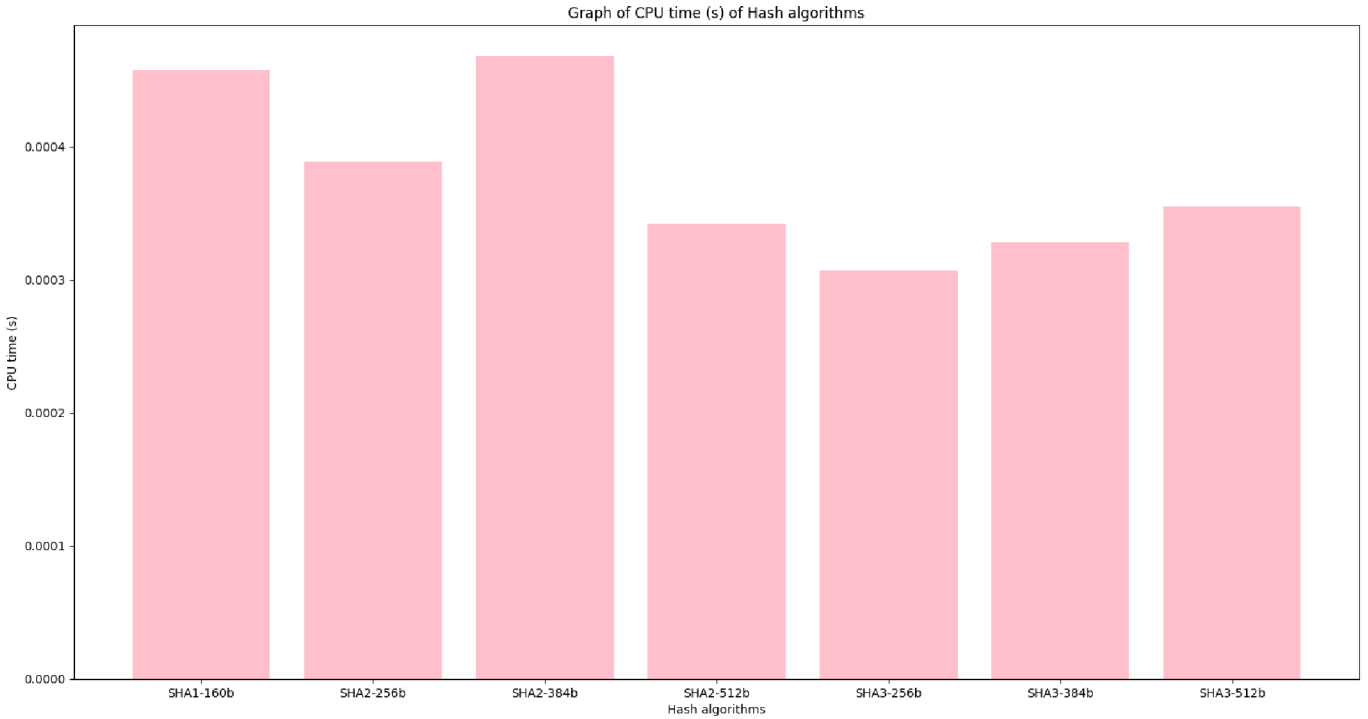
K testování byla dodána také platforma Raspberry Pi Zero W [53], jež nebyla zmíněna v kapitole 5.2 a nebyla testována. Důvodem k netestování byla značná časová náročnost a absence konektorů potřebných k ovládání přístroje pomocí počítačových periferií, které byly k dispozici.



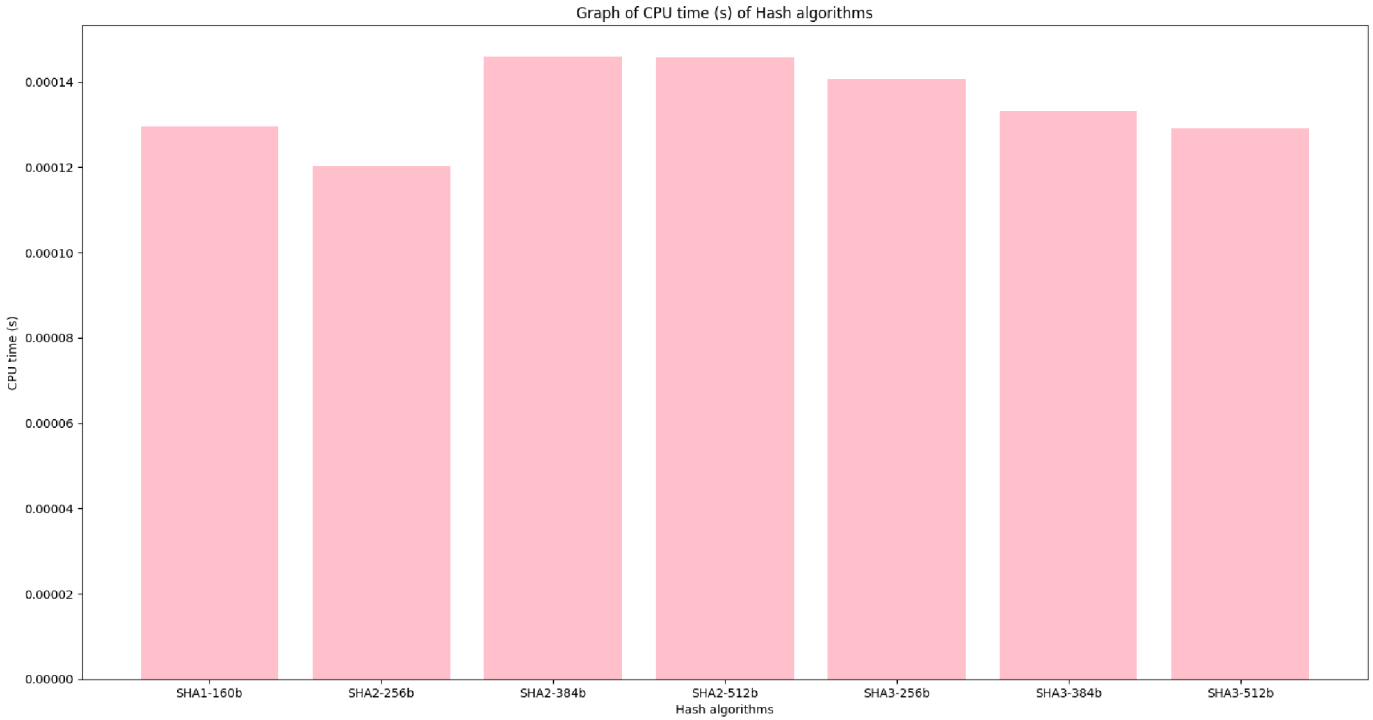
Obr. 12.1: Doba provedení hashovacích algoritmů na Raspberry Pi B.



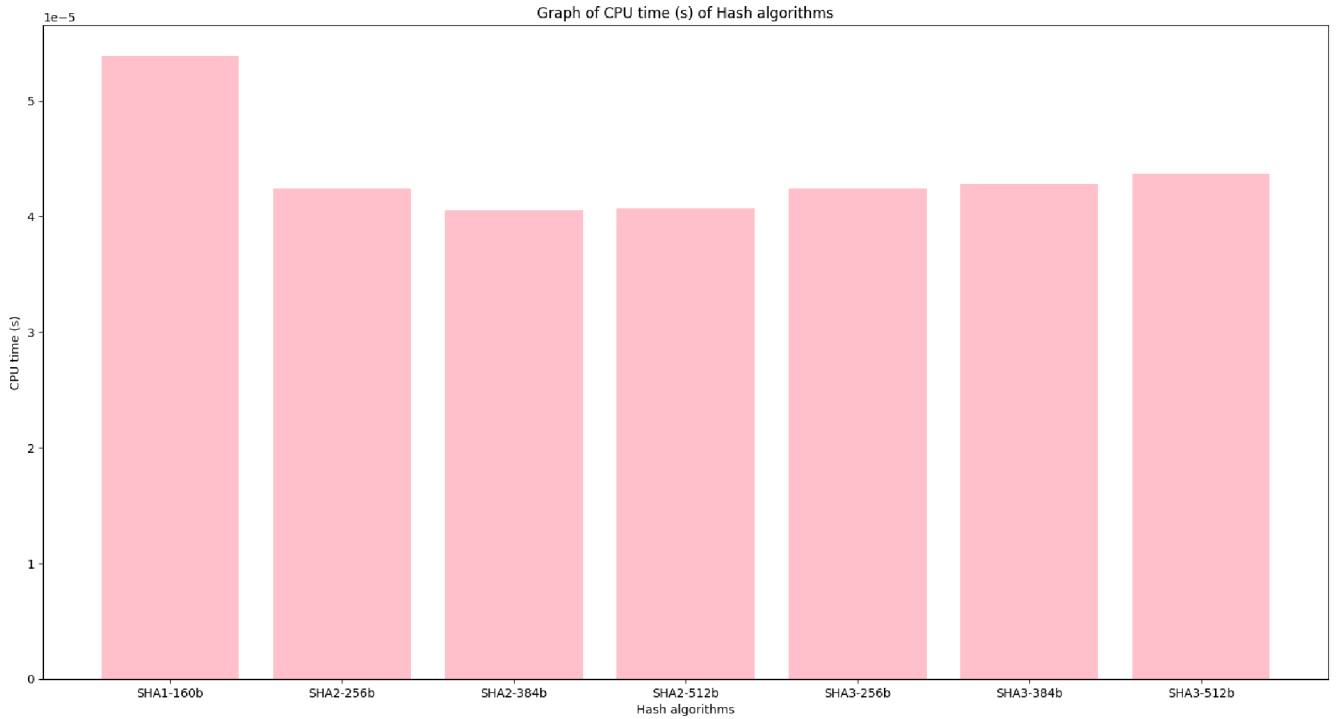
Obr. 12.2: Doba provedení hashovacích algoritmů na Raspberry Pi 2.



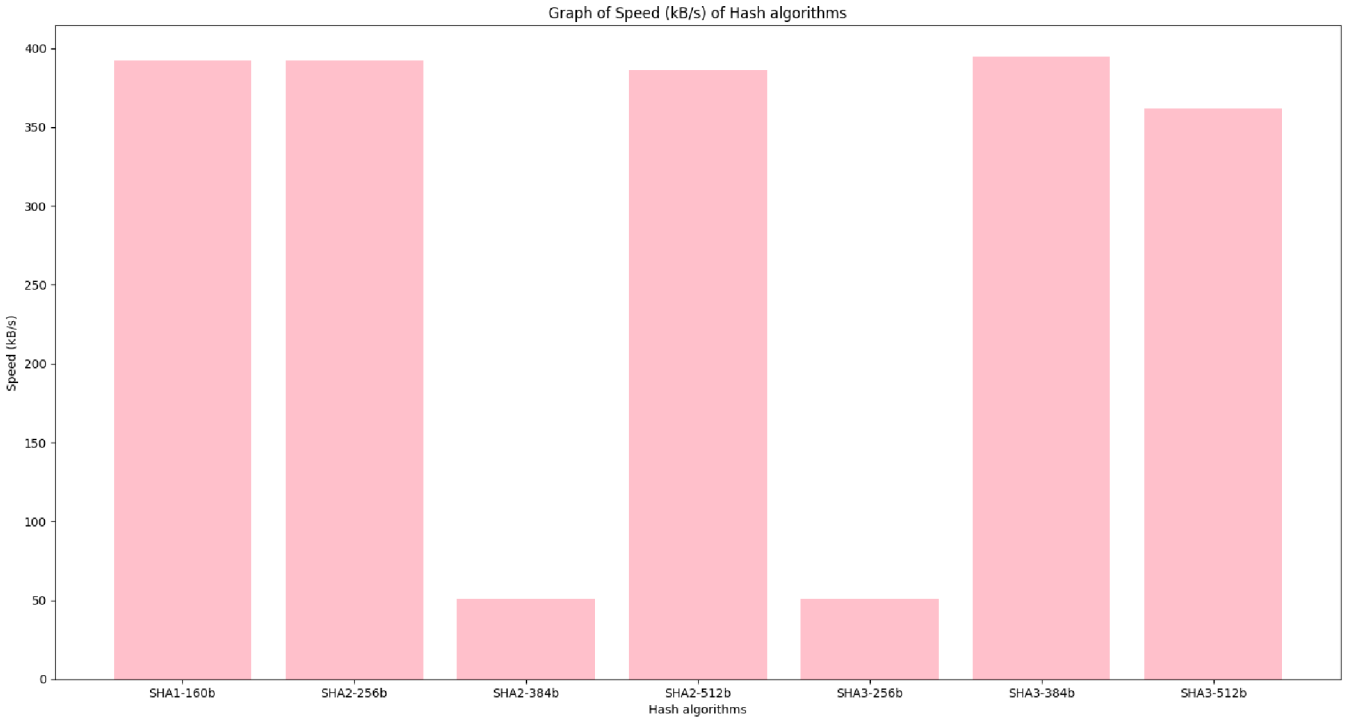
Obr. 12.3: Doba provedení hashovacích algoritmů na Raspberry Pi 3.



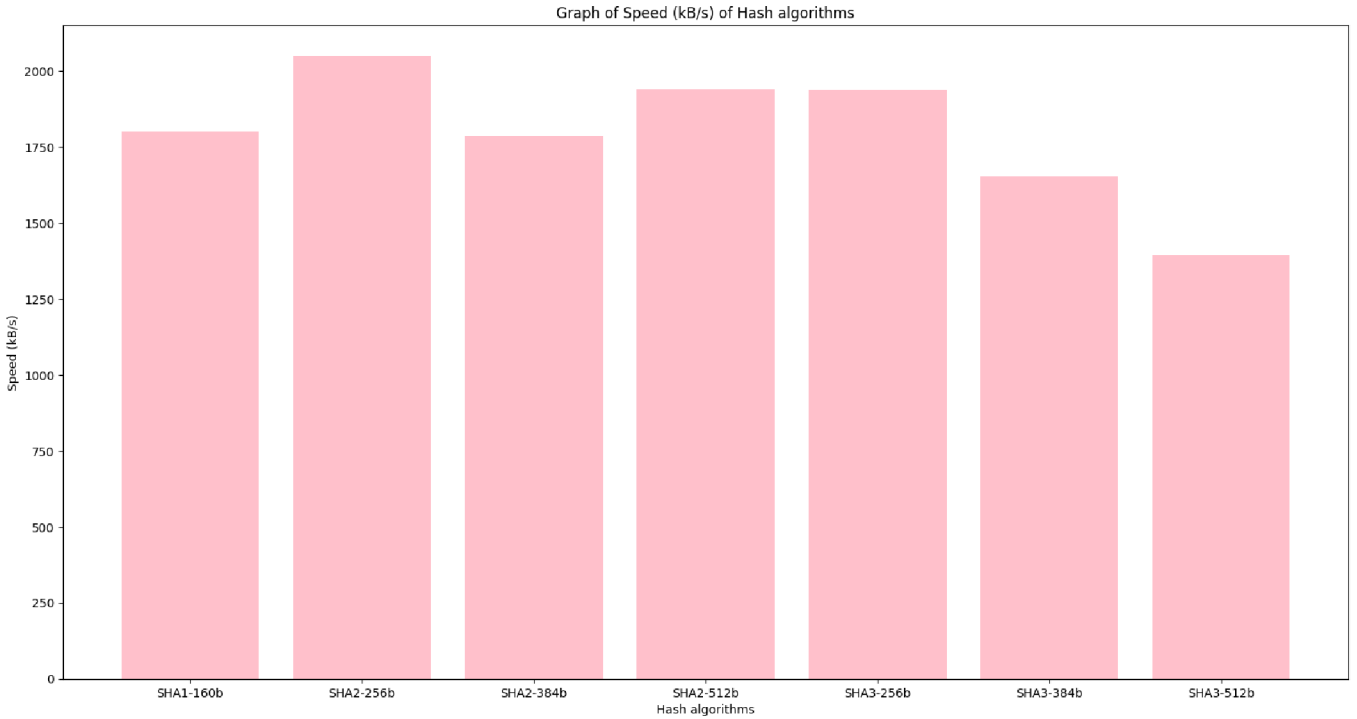
Obr. 12.4: Doba provedení hashovacích algoritmů na Raspberry Pi 4.



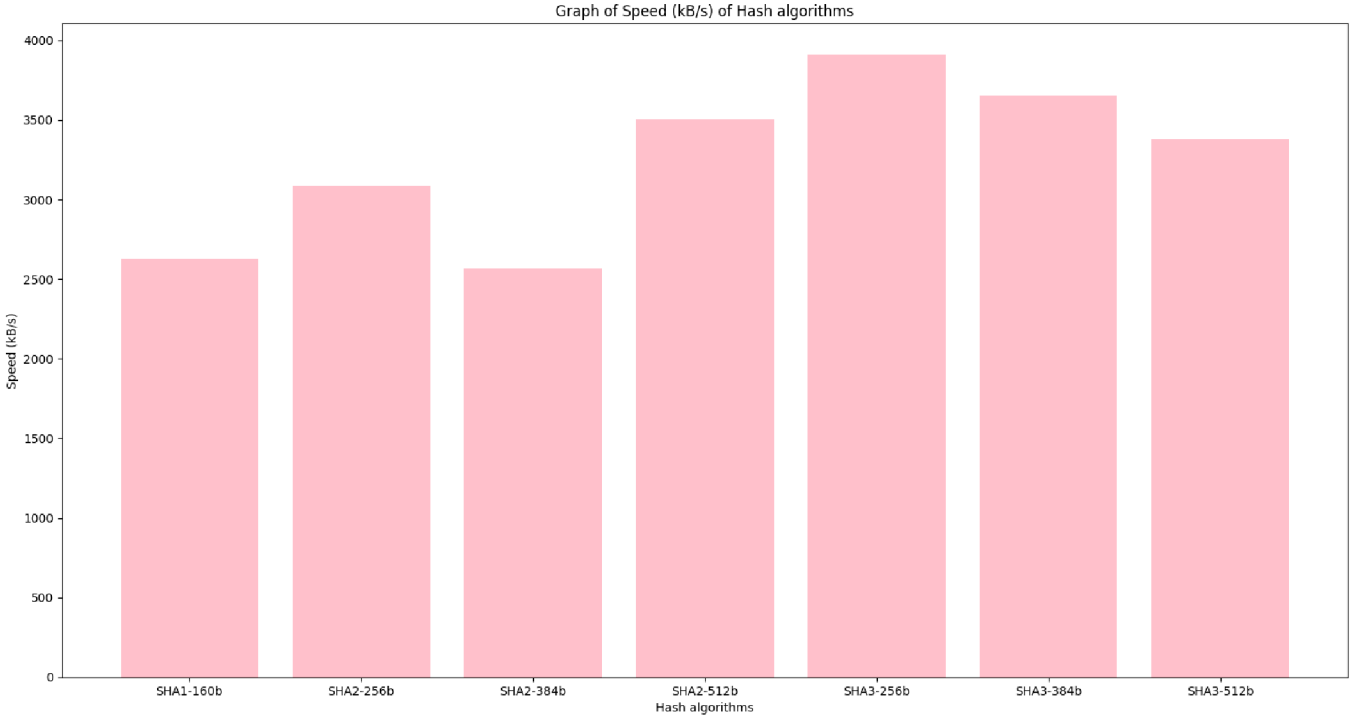
Obr. 12.5: Doba provedení hashovacích algoritmů na stolním PC.



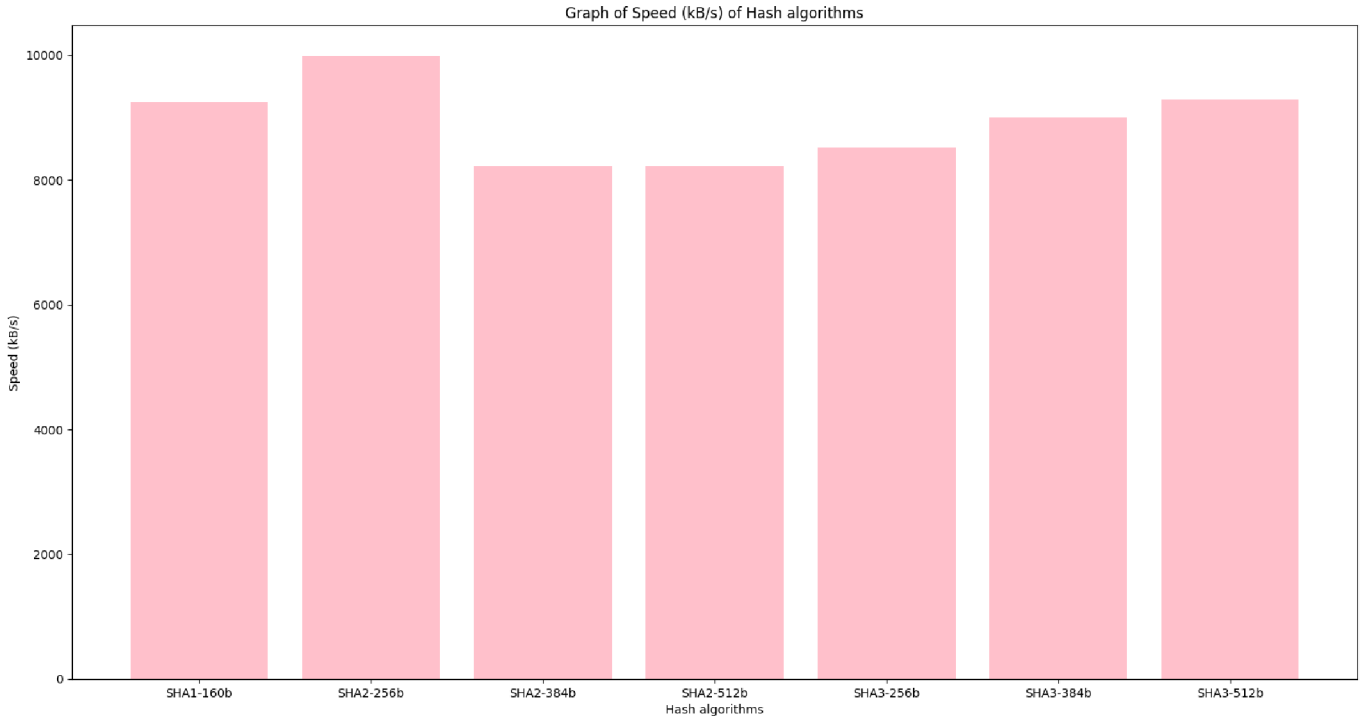
Obr. 12.6: Rychlost provedení hashovacích algoritmů na Raspberry Pi B.



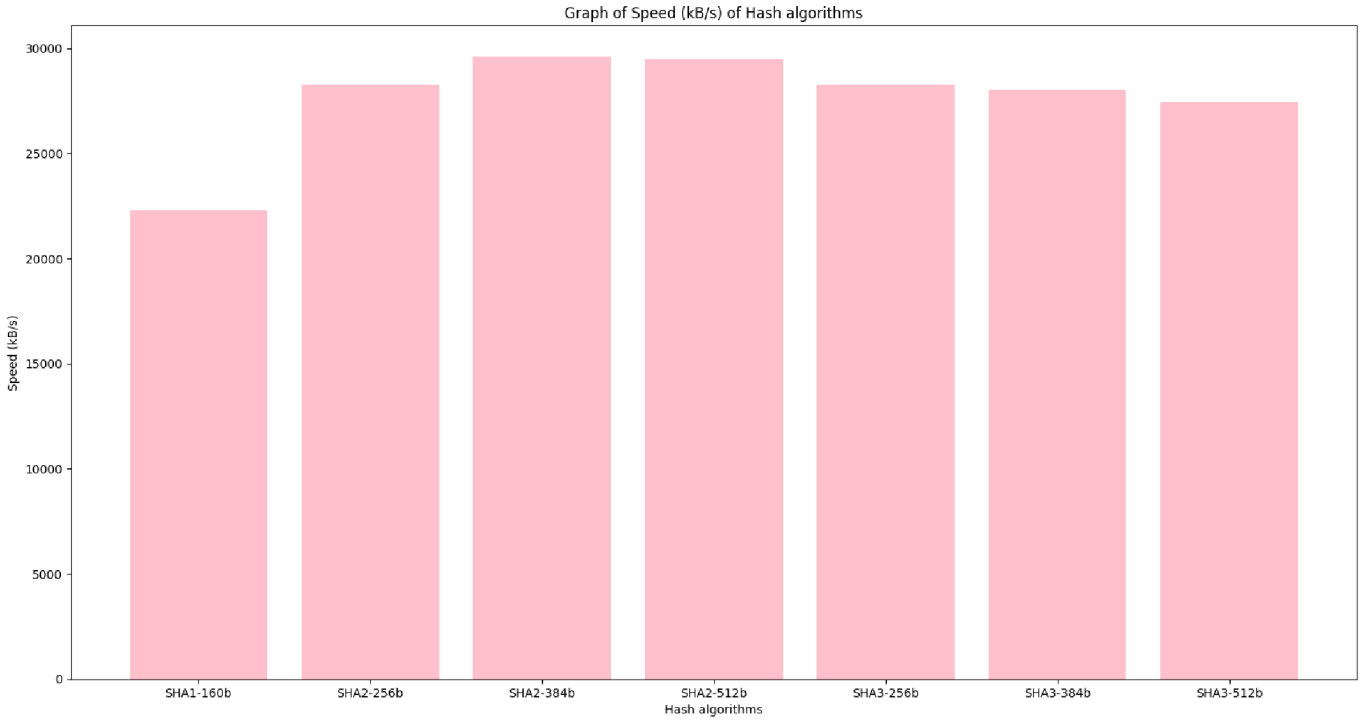
Obr. 12.7: Rychlost provedení hashovacích algoritmů na Raspberry Pi 2.



Obr. 12.8: Rychlost provedení hashovacích algoritmů na Raspberry Pi 3.



Obr. 12.9: Rychlost provedení hashovacích algoritmů na Raspberry Pi 4.



Obr. 12.10: Rychlost provedení hashovacích algoritmů na stolním PC.

Závěr

Tato diplomová práce se zabývala rešerší kryptografických primitiv a platforem využívaných v energetice, po které následovalo vytvoření nástroje pro otestování vybraných primitiv na vybraných embedded platformách.

Úvod práce se věnuje základnímu popisu obecně využívaných kryptografických primitiv zajišťujících bezpečnostní požadavky doby pro komunikaci výpočetních zařízení a ukládání dat. Ve druhé kapitole je krátce zmíněn známý protokol DLMS/COSEM využívaný systémy provozovanými v energetice pro příklad využití kryptografie na embedded platformách. Následující kapitola rozebírá bezpečnostní mechanismy užívané tímto protokolem, které cílí na zachování bezpečnosti komunikujících systémů. V návaznosti na bezpečnost těchto systémů jsou následně zmíněny instituce, na jejichž základě jsou poté určeny požadavky kryptografických primitiv na provoz embedded platforem v energetice. Součástí těchto požadavků jsou také detailnější popisy již zmíněných kryptografických funkcí.

Závěr teoretické části práce je věnován produktům společnosti Raspberry Pi, které byly použity pro testování vybraných kryptografických prostředků. Dále jsou zmíněny také vybrané kryptografické knihovny poskytující nástroje na provádění kryptografických operací na Raspberry Pi platformách. Poslední kapitola této části práce se věnuje krátkému testování některých kryptografických primitiv na platformě Raspberry Pi 3 za pomoci softwarového nástroje OpenSSL speed.

Praktická část této práce začíná stanovením teoretických požadavků na výsledný testovací nástroj. Následuje kapitola obsahující diskuse a odůvodnění výběru jazyka Python a vývojářské aplikace MS Visual Code.

Největším přínosem této práce je kapitola pojednávají o tvorbě nástroje CipherBench. V rámci dílčích podkapitol této části jsou rozebrány způsoby implementace algoritmů a měřících funkcí společně se systémem ukládání výsledků a možnostmi jejich prezentace. Navazující kapitola ukazuje způsoby správného spuštění aplikace včetně popisu samotné adresářové struktury aplikace.

Závěr diplomové práce je věnován analýze a diskusi naměřených výsledků na platformách Raspberry Pi a vlastním stolním počítači. Naměřené výsledky platforem jsou porovnávány pouze v jedné kategorii implementovaných primitiv z důvodu objemu naměřených dat, avšak veškerá naměřená data jsou obsahem elektronické přílohy práce, stejně jako výsledná aplikace.

Literatura

[1] BRYCHTA, Josef. *Srovnání kryptografických primitiv využívajících eliptických křivek na různých hardwarových platformách.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Radek Fujdiak, Ph.D.

[2] KOHOUT, David. *Zátěžový generátor zpráv DLMS/COSEM.* Brno, 2019. Bakalářská práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Tomáš Lieskovan.

[3] *Cryptography.* TechTarget Search Security [online]. Newton, USA: TechTarget, c2000-2022 [cit. 2022-05-21]. Dostupné z: https://www.techtarget.com/searchsecurity/definition/cryptography

[4] *Příklady fungování kryptografických algoritmů v digitálním podpisu,* MENEZES, Alfred J., Paul C. van OORSCHOT a Scott A. VANSTONE. Digital Signature. Handbook of applied cryptography. 5th ed. Boca Raton: CRC Press, 1997, s. 425-481. ISBN 0849385237.

[5] KARÁSEK, Jan. *Hashovací funkce - charakteristika, implementace a kolize.* Brno, 2009. Diplomová práce. Vysoké učení technické v Brně. Vedoucí práce Ing. Petra Lambertová.

[6] Green Book Ed.9: *DLMS/COSEM Architecture and Protocols* [online]. Zug, Switzerland: DLMS User Association, 2019 [cit. 2021-12-02]. Dostupné z: https://www.dlms.com/files/Green_Book_Edition_9-Excerpt.pdf

[7] *A White Paper by the DLMS User Association.* DLMS: Device Languafe Message Specification [online]. Steinhausen, Switzerland: DLMS User Association, c1997-2019 [cit. 2021-12-06]. Dostupné z: https://www.dlms.com/files/DLMS-White-Paper-Security_November_2019.pdf

[8] *National Institute od Standards and Technology* [online]. Gaithersbur, USA: U.S. Department of Commerce, c2021 [cit. 2022-05-21]. Dostupné z: https://www.nist.gov/

[9] *European Union Agency for Cybersecurity* [online]. Athény, Řecko: European Union Agency for Cybersecurity, c2005-2022 [cit. 2022-05-21]. Dostupné z: https://www.enisa.europa.eu/

[10] BARKER, Elaine a Allen ROGINSKY. *Transitioning the Use of Cryptographic Algorithms and Key Lengths* [online]. , 0-33 [cit. 2021-12-02]. Dostupné z: doi: 10.6028/NIST.SP.800-53r5

[11] *Digital Signature Standard (DSS)* [online]. 2013, , 0-130 [cit. 2021-12-02]. Dostupné z: doi: 10.6028/NIST.FIPS.186-4

[12] *Secure Hash Standard (SHS)* [online]. 2015, , 0-36 [cit. 2021-12-02]. Dostupné z: doi: 10.6028/NIST.FIPS.180-4

[13] BARKER, Elaine a John KELSEY. *Recommendation for Random Number Generation Using Deterministic Random Bit Generators.* [online]. 2015, , 13-60 [cit. 2021-12-02]. Dostupné z: doi: 10.6028/NIST.SP.800-90Ar1

[14] *Enisa algorithm, key sizes and parameters report.* ENISA [online]. Iraklio, Řecko: European Union Agency for Cybersecurity, c2005-2021 [cit. 2021-12-06]. Dostupné z: https://www.enisa.europa.eu/publications/algorithms-key-sizes-and-parameters-report

[15] *Recommended cryptographic measures - Securing Personal Data.* ENISA [online]. Iraklio, Řecko: European Union Agency for Cybersecurity, c2005-2021 [cit. 2021-12-06]. Dostupné z: https://www.enisa.europa.eu/publications/recommended-cryptographic-measures-securing-personal-data

[16] *Minimální požadavky na kryptografické algoritmy.* [online]. 2018, , 0-8 [cit. 2021-12-02]. Dostupné z: https://www.nukib.cz/cs/infoservis/doporuceni/1504-doporuceni-v-oblasti-kryptografickych-prostredku/

[17] *Nationa Cyber Security Centre* [online]. Londýn, UK: GOV.UK, c2016-2022 [cit. 2022-05-21]. Dostupné z: https://www.ncsc.gov.uk/

[18] *Crypto competitions: AES: the Advanced Encryption Standard* [online]. Gaithersbur, USA: U.S. Department of Commerce, c2013-2017 [cit. 2022-05-21]. Dostupné z: https://competitions.cr.yp.to/aes.html

[19] *Stream Ciphers.* GeeksforGeeks [online]. Noida, Indie: geeksforgeeks, c2015-2022, 16.10.2020 [cit. 2022-05-21]. Dostupné z: https://www.geeksforgeeks.org/stream-ciphers/

[20] *Five DRBG Algorithms Based on Hash Functions and Block Cipher.* National Institute of Standards and Technology [online]. Gaithersburg, USA: U.S. Department of Commerce, c1999-2021, 2004 [cit. 2022-05-21]. Dostupné z: https://csrc.nist.gov/CSRC/media/Events/Random-Number-Generation-Workshop-2004/documents/HashBlockCipherDRBG.pdf

[21] *Embedded system.* Omnisci.com [online]. Omnisci, c2021 [cit. 2021-12-06]. Dostupné z: https://www.omnisci.com/technical-glossary/embedded-systems

[22] *Raspberry Pi Products.* Raspberry Pi [online]. Cambridge, Spojené Království: Raspberry Pi Foundation, c2008-2021 [cit. 2021-12-06]. Dostupné z: https://www.raspberrypi.com/products/

[23] *Minipočítače Raspberry Pi.* RPishop.cz [online]. RPishop.cz, c2021 [cit. 2021-12-06]. Dostupné z: https://rpishop.cz/249-raspberry-pi

[24] *Raspberry Pi - Model B.* SparkFun Electronics [online]. Colorado, USA: RPishop.cz, c2014 [cit. 2021-12-06]. Dostupné z: https://www.sparkfun.com/products/retired/11546

[25] *OpenSSL ciphers.* OpenSSL [online]. The OpenSSL Project Authors, c1999-2021 [cit. 2021-12-06]. Dostupné z: https://www.openssl.org/docs/man1.1.1/man1/ciphers.html

[26] *OpenSSL speed.* OpenSSL [online]. The OpenSSL Project Authors, c1999-2021 [cit. 2021-12-06]. Dostupné z: https://www.openssl.org/docs/man1.1.1/man1/openssl-speed.html

[27] *OpenSSL: Cryptography and SSL/TLS toolkit* [online]. The OpenSSL Project Authors, c2021 [cit. 2022-05-17]. Dostupné z: https://www.openssl.org/

[28] *Crypto++ Library 8.6* [online]. Crypto++ community, c2014 [cit. 2021-12-06]. Dostupné z: https://www.cryptopp.com/

[29] *WolfCrypt Embedded Crypto Engine.* WolfSSL [online]. wolfSSL, c2021 [cit. 2021-12-06]. Dostupné z: https://www.wolfssl.com/products/wolfcrypt-2/

[30] *Libgcrypt.* GnuPG [online]. The Gnu Project, c1998-2020 [cit. 2021-12-06]. Dostupné z: https://gnupg.org/software/libgcrypt/index.html

[31] *LibreSSL* [online]. OpenBSD Foundation, c2014-2021 [cit. 2021-12-06]. Dostupné z: libressl.org

[32] *Operating system images.* Raspberry Pi [online]. Anglie, Wales: Raspberry Pi Foundation, c2009 [cit. 2022-05-17]. Dostupné z: https://www.raspberrypi.com/software/operating-systems/

[33] *Raspberry Pi Documentation.* Raspberry Pi [online]. Anglie, Wales: Raspberry Pi Foundation, c2009 [cit. 2022-05-17]. Dostupné z: https://www.raspberrypi.com/documentation/computers/getting-started.html

[34] *Raspberry Pi Foundation* [online]. Anglie, Wales: Raspberry Pi Foundation, c2022 [cit. 2022-05-17]. Dostupné z: https://www.raspberrypi.org/

[35] *Pyca/cryptography* [online]. Individual Contributors, c2021 [cit. 2022-05-17]. Dostupné z: https://cryptography.io/en/latest/

[36] *Loxodo/twofish.py*. In: GitHub [online]. San Francisco, USA: GitHub, 2018 [cit. 2022-05-17]. Dostupné z: https://github.com/sommer/loxodo/blob/master/src/twofish/twofish.py

[37] *MESNIL, Cedric. Source code for ecpy.ecschnorr*. ECPy's documentation [online]. Cedric Mensil, 2016 [cit. 2022-05-17]. Dostupné z: https://ecpy.readthedocs.io/en/latest/_modules/ecpy/ecschnorr.html

[38] *Fpgaminer/python-hmac-drbg*. In: GitHub [online]. San Francisco, USA: GitHub, 2015 [cit. 2022-05-17]. Dostupné z: https://github.com/fpgaminer/python-hmac-drbg/blob/master/hmac_drbg/hmac_drbg.py

[39] *GitHub* [online]. San Francisco, USA: GitHub, c2022 [cit. 2022-05-17]. Dostupné z: https://github.com/

[40] *Pyserpent/serpent.py*. In: GitHub [online]. San Francisco, USA: GitHub, 2017 [cit. 2022-05-17]. Dostupné z: https://github.com/sysopfb/pyserpent/blob/master/serpent.py

[41] *Ctypes - A foreign function library for Python*. Python.org [online]. Wilmington, Delaware, USA: Python Software Foundation, c2022 [cit. 2022-05-17]. Dostupné z: https://docs.python.org/3/library/ctypes.html

[42] *Time - Time access and conversions*. Python.org [online]. Wilmington, Delaware, USA: Python Software Foundation, c2022 [cit. 2022-05-17]. Dostupné z: https://docs.python.org/3/library/time.html

[43] *Memory-profiler*. PyPI [online]. San Francisco, USA: Python Software Foundation, c2022 [cit. 2022-05-17]. Dostupné z: https://pypi.org/project/memory-profiler/

[44] *Tracemalloc — Trace memory allocations*. Python.org [online]. Wilmington, Delaware, USA: Python Software Foundation, c2022 [cit. 2022-05-17]. Dostupné z: https://docs.python.org/3/library/tracemalloc.html

[45] *Powerapi-ng/pyRAPL*. In: GitHub [online]. San Francisco, USA: GitHub, c2022 [cit. 2022-05-17]. Dostupné z: https://github.com/powerapi-ng/pyRAPL

[46] *Energyusage*. PyPI [online]. San Francisco, USA: Python Software Foundation, c2022 [cit. 2022-05-17]. Dostupné z: https://pypi.org/project/energyusage/

[47] *Xml.dom.minidom — Minimal DOM implementation.* Python.org [online]. Wilmington, Delaware, USA: Python Software Foundation, 2022 [cit. 2022-05-17]. Dostupné z: https://docs.python.org/3/library/xml.dom.minidom.html

[48] *Qt Designer Manual.* Qt [online]. Espoo, Finsko: Qt Company, 2022 [cit. 2022-05-17]. Dostupné z: https://doc.qt.io/qt-5/qtdesigner-manual.html

[49] *Qt 5.15.* Qt [online]. Espoo, Finsko: Qt Company, c2022 [cit. 2022-05-17]. Dostupné z: https://doc.qt.io/qt-5.15/

[50] *Matplotlib - Visualization with Python* [online]. The Matplotlib Development team, c2021 [cit. 2022-05-17]. Dostupné z: https://matplotlib.org/

[51] *ECPy's documentation* [online]. Francie: Cedric Mesnil, 2016 [cit. 2022-05-17]. Dostupné z: https://ec-python.readthedocs.io/en/latest/

[52] *Eciespy.* PyPI [online]. San Francisco, USA: Python Software Foundation, 2022 [cit. 2022-05-17]. Dostupné z: https://pypi.org/project/eciespy/s

[53] *Raspberry Pi Zero W.* RPishop.cz [online]. RPishop.cz, c2022 [cit. 2022-05-18]. Dostupné z: https://rpishop.cz/zero/647-raspberry-pi-zero-w-4053199547425.html

Seznam symbolů a zkratek

DLMS	Jazykový standard pro chytrá zařízení - Device Language Message specification
COSEM	Model rozhraní pro komunikaci zařízení sloužící k měření v energetice - Companion Specification for Metering Energy
UA	Společenství uživatelů - User Association
OSI/ISO	Referenční komunikační model systému - Open System Interconnection Reference Model
NIST	Americký národní institut standardů a techniky - National Institute of Standards and Technology
ENISA	Evropská agentura pro bezpečnost sítí a informací - European Union Agency for Cybersecurity
NÚKIB	Národní úřad pro kybernetickou a informační bezpečnost
NCSC	Národní centrum kybernetické bezpečnosti Velké Británie - National Cyber Security Centre
AES	Standardizovaný algoritmus šifrování dat - Advanced Encryption Standard
3DES	Bloková šifra založená na šifrovacím algoritmu DES - Data Encryption Standard
XOR	Logická operace exkluzivní disjunkce
SHA	Rozšířená hashovací funkce - Secure Hash Algorithm
MD	Rodina hashovacích funkcí - Message Digest
DH	Kryptografický protokol Diffie-Hellman výměny klíčů - Diffie-Hellman key exchange
KEM	Mechanismus zapouzdření klíčů - Key Encapsulation Mechanism
SoC	Integrovaný obvod, nebo také systém na čipu - System on a Chip
CPU	Centrální procesorová jednotka - Central Processing Unit
GPU	Grafický procesor - Graphic Processing Unit

HDMI	Označení nekomprimovného obrazového a zvukového signálu v digitální podobě - High-Definition Multimedia Interface
USB	Univerzální sériová sběrnice - Universal Serial Bus
SSL	Protokol poskytující zabezpečení komunikace šifrováním a autentizací - Secure Socket Layer
TLS	Následník protokolu TLS poskytující zabezpečení komunikace - Transport Layer Security
KEM	Mechanismus zapouzdření klíče - Key Encapsulation Mechanism
GUI	Uživatelské rozhraní - Graphic User Interface

Seznam příloh

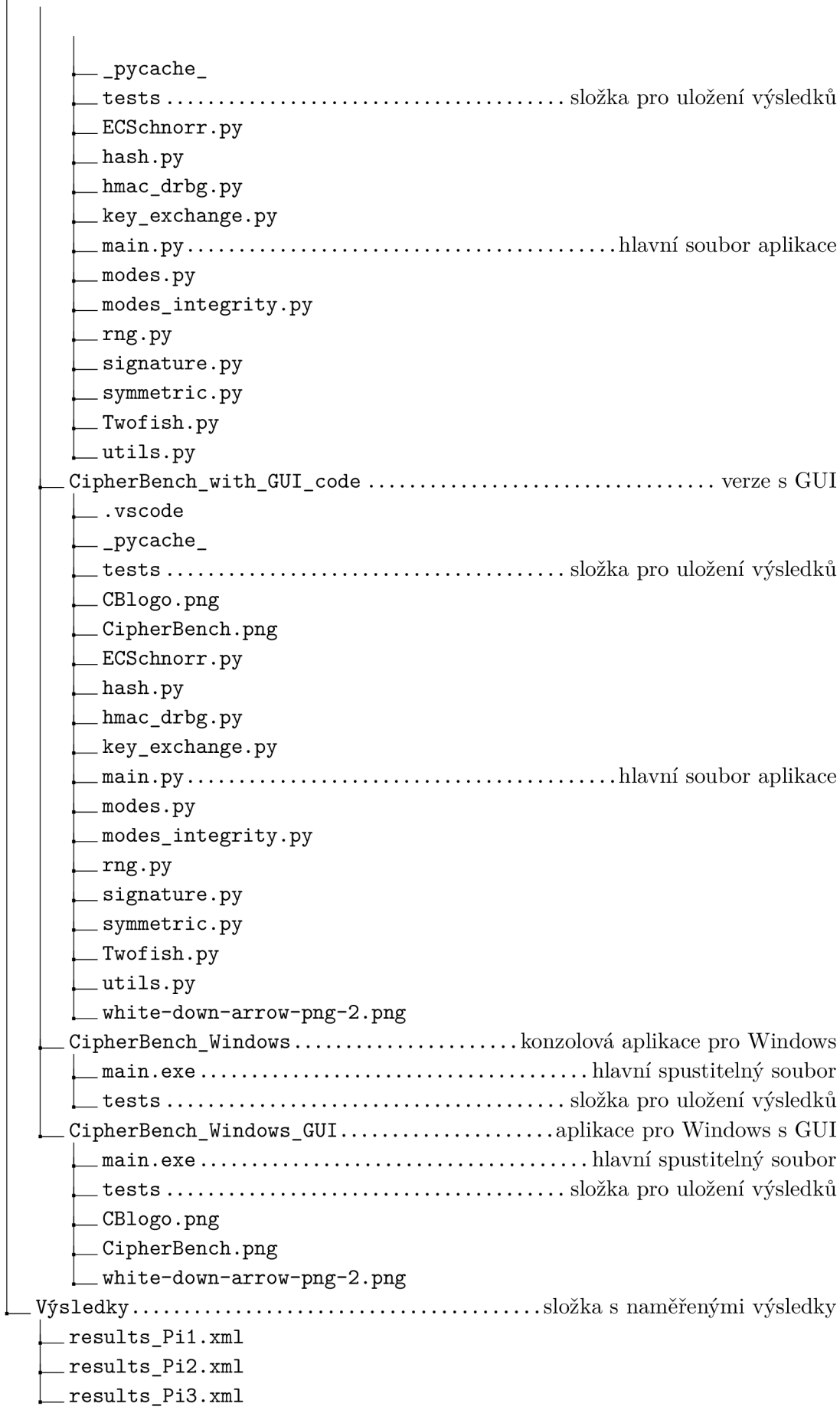
A Obsah elektronické přílohy

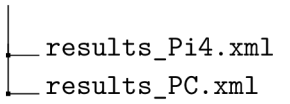
Obsahem elektronické přílohy jsou všechny vytvořené verze aplikace CipherBench napsané v jazyce Python (konkrétněji se jedná o 64 bitovou verzi interpreta 3.9.7) pro operační systém Linux. V případě těchto aplikací se jedná o zdrojové soubory jednotlivých kryptografických primitiv, či jejich implementací skrze knihovny. Hlavním souborem určeným ke spuštění této aplikace ke soubor *main.py* (případně soubor *CipherBench.py*). K těmto verzím aplikace se pojí složka *lib* obsahující příklad souborů externích knihoven sestavených na Raspberry Pi 4 pro přiložení k hlavním souborům aplikace v případě požadavku na spuštění aplikace bez nutnosti instalace těchto knihoven na testovaný systém.

Současně s těmito verzemi jsou přiloženy složky obsahující spustitelnou verzi (soubory s příponou `exe`) této aplikace na operačním systému Windows.

Složka *Výsledky* obsahuje naměřená data pomocí této aplikace na zařízeních Raspberry Pi a vlastního počítače.

```
/ ..... kořenový adresář přiloženého archivu
├── lib ..... úložiště příkladu vkládaných knihoven
│   ├── ecies ..... knihovna ecies
│   ├── eciespy-0.3.11.dist-info ..... složka s informacemi o knihovně ecies
│   ├── ecpy ..... knihovna ECPy
│   ├── ECPy-1.2.5.dist-info ..... složka s informacemi o knihovně ECPy
│   ├── matplotlib ..... knihovna matplotlib
│   ├── matplotlib-3.5.2.dist-info ... složka s informacemi o knihovně matplotlib
│   └── mpl_toolkits ..... pomocná knihovna ke knihovně matplotlib
└── verze ..... úložiště verzí aplikace
    ├── CipherBench_cmd_with_colors_code ..... barevná konzole
    │   ├── .vscode
    │   ├── _pycache_
    │   ├── tests ..... složka pro uložení výsledků
    │   ├── ECSchnorr.py
    │   ├── hash.py
    │   ├── hmac_drbg.py
    │   ├── key_exchange.py
    │   ├── main.py ..... hlavní soubor aplikace
    │   ├── modes.py
    │   ├── modes_integrity.py
    │   ├── rng.py
    │   ├── signature.py
    │   ├── symmetric.py
    │   ├── Twofish.py
    │   └── utils.py
    └── CipherBench_cmd_without_colors_code ..... černobílá konzole
        └── .vscode
```


pycache
tests ..... složka pro uložení výsledků
ECSchnorr.py
hash.py
hmac_drbg.py
key_exchange.py
main.py ..... hlavní soubor aplikace
modes.py
modes_integrity.py
rng.py
signature.py
symmetric.py
Twofish.py
utils.py
CipherBench_with_GUI_code ..... verze s GUI
.vscode
pycache
tests ..... složka pro uložení výsledků
CBlogo.png
CipherBench.png
ECSchnorr.py
hash.py
hmac_drbg.py
key_exchange.py
main.py ..... hlavní soubor aplikace
modes.py
modes_integrity.py
rng.py
signature.py
symmetric.py
Twofish.py
utils.py
white-down-arrow-png-2.png
CipherBench_Windows ..... konzolová aplikace pro Windows
main.exe ..... hlavní spustitelný soubor
tests ..... složka pro uložení výsledků
CipherBench_Windows_GUI ..... aplikace pro Windows s GUI
main.exe ..... hlavní spustitelný soubor
tests ..... složka pro uložení výsledků
CBlogo.png
CipherBench.png
white-down-arrow-png-2.png
Výsledky ..... složka s naměřenými výsledky
results_Pi1.xml
results_Pi2.xml
results_Pi3.xml


results_Pi4.xml
results_PC.xml