VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2022 Tomáš Škurla

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

POROVNÁNÍ KRYPTOGRAFICKÉ VÝKONNOSTI MIKROKONTROLERŮ ŘADY ARM CORTEX-A

COMPARISON OF CRYPTOGRAPHY PERFORMANCE OF MICROCONTROLLERS ARM CORTEX-A

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Tomáš Škurla

AUTHOR

VEDOUCÍ PRÁCE — doc. Mgr. Karel Slavíček, Ph.D.

SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Telekomunikační a informační systémy**

Ústav telekomunikací

Student: Tomáš Škurla ***ID:*** 219289

Ročník: 3 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Porovnání kryptografické výkonnosti mikrokontrolerů řady ARM Cortex-A

POKYNY PRO VYPRACOVÁNÍ:

Cílem práce je zpracovat přehled vlastností mikrokontrolerů řady ARM Cortex-A z pohledu jejich hardwarové podpory kryptografického zabezpečení komunikace a navrhnout metodiku praktického porovnání výkonnosti dostupných mikrokontrolerů založených na této architektuře, včetně případných hardwarových přípravků, budou-li potřeba. V praktické části práce - navazující BP - bude provedeno reálné měření dle této metodiky.

DOPORUČENÁ LITERATURA:

[1] ARM® Cortex™-A Series Programmer's Guide. Dostupné on-line: https://developer.arm.com/documentation/den0013/latest

[2] Vanderbauwhede, W., & Singer, J. (2019). Operating systems foundations with Linux on the Raspberry Pi: Textbook. Arm Education Media.

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 31.5.2022

Vedoucí práce: doc. Mgr. Karel Slavíček, Ph.D.

prof. Ing. Jiří Mišurec, CSc.
předseda rady studijního programu

Abstrakt

Bakalárska práca sa zaoberá meraním kryptografického výkonu mikrokontrolérov série ARM Cortex-A. Teoretická časť sa sústreďuje na opis kryptografie, pričom zároveň vysvetľuje princípy fungovania jednotlivých kryptografických systémov. Ďalej je popísané fungovanie ARM procesorov, ich historický vývin, podpora pre šifrovanie a zistenie jej podpory v rámci procesorov. Nasleduje definovanie metrík pre testovanie, ktorého častou je i vymenovanie použitého hardwaru pri procese testovania, pričom hlavným zameraním, je podpora akcelerácie kryptografie a šifrovania. Súčasťou je aj opis plánovaného prostredia a softwaru použitého v rámci testovania. Praktická časť zahrňuje opis reálnej prípravy nadefinovaného prostredia a jeho prípadných úprav v rámci jednotlivých potrieb testovania. Záverom práce je vyhodnotenie spracovaných údajov a následné vyvodenie záverov.

Klíčová slova

ARM, Cortex-A, Kryptografia, Bloková šifra, Prevádzkové režimy blokových šifier, AES, SoC, Numpy, Pandas, Matplotlib, Testovanie

Abstract

The bachelor thesis deals with the measurement of cryptographic performance of ARM Cortex-A series microcontrollers. The theoretical part focuses on the description of concepts of cryptography, while at the same time explaining the principles of operation of individual cryptographic systems. Afterwards, thesis also describes the inner workings of ARM processors, their historical development, support for cryptography and detection of said support within processors. This is followed by defining the metrics for testing, part of which includes enumerating the hardware used in the testing process, with the focus being on the cryptography acceleration and encryption support. This also includes a description of the planned environment and software used in the testing. The practical part includes a description of the actual preparation of the defined environment and its possible modifications within the individual testing needs. The thesis concludes with an evaluation of the processed data and subsequent drawing of conclusions.

Keywords

ARM, Cortex-A, Cryptography, Block Cypher, Block cipher modes of operation, AES, SoC, Numpy, Pandas, Matplotlib, Testing

Bibliografická citace

ŠKURLA, Tomáš. *Porovnání kryptografické výkonnosti mikrokontrolerů řady ARM Cortex-A*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2022. 54 s. Bakalářská práce. Vedoucí práce: doc. Mgr. Karel Slavíček, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení studenta:	*Tomáš Škurla*
VUT ID studenta:	*219289*
Typ práce:	*Bakalářská práce*
Akademický rok:	*2021/22*
Téma závěrečné práce:	*Porovnání kryptografické výkonnosti mikrokontrolerů řady ARM Cortex-A*

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 jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

.

V Brně dne: 11. května 2022 ----------

podpis autora

Poďakovanie

Týmto spôsobom by som chcel poďakovať môjmu vedúcemu bakalárskej práce pánovi Doc. Mgr. Karlovi Slavíčkovi, Ph.D. za účinnú metodickú a odbornú pomoc pri vytváraní mojej bakalárskej práce. Ďalej by som chcel poďakovať rodine za podporu a trpezlivosť behom môjho štúdia.

V Brně dne: 11. května 2022 ----------

podpis autora

Obsah

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

1. Úvod

V dnešnej dobe sa stáva kryptografia a bezpečnosť v rámci internetovej komunikácie stále viacej potrebnejšia. Toto nie je pravda len v prípade domácich počítačov, ale i v rámci mobilov, tabletov či dokonca techniky na spracovávanie multimédií alebo superpočítačov používaných v technologickom výskume.

Moja práca sa zaoberá porovnaním kryptografickej výkonnosti najčastejšie používaných ARM procesorov z rodiny Cortex-A. Prácu je možno obsahovo rozdeliť do šiestich kapitol, pričom kapitoly jedna až štyri popisujú teoretickú časť práce a kapitoly päť až šesť zasa časť praktickú. Prvá kapitola sa zaoberá úvodom do kryptografie, vysvetlenie jej základných poznatkov a následné rozdelenie podľa použitých šifrovacích systémov. Ďalšie časti tejto kapitoly vysvetľujú fungovanie šifrovacieho algoritmu AES.

Druhá kapitola sa zaoberá technológiou ARM procesorov. Obsahom je ukážka historického vývoja procesora a inštrukčnej sady, zobrazenie a popis jej architektúry. Ďalšia podkapitola sa zaoberá aj porovnaním so stavbou procesoru x86 a overením existencie hardwarového rozšírenia kryptografickej akcelerácie v Unix-like operačných systémoch.

Tretia kapitola opisuje hardware použitý v testovaní, popisuje vlastnosti jednotlivých vzoriek v rámci hardwarovej výbavy a zároveň definuje postup a metriky použité v rámci testovania vykonaného v praktickej časti práce.

Štvrtá kapitola popisuje výber softwaru ktorý je použitý v testovaní, objasňuje jeho jednotlivé vlastnosti a fungovanie. Súčasťou kapitoly je taktiež krátke vysvetlenie dôvodu pre vyber daného softwaru.

Piata kapitola a prvá časť praktickej práce opisuje kroky uskutočnené v rámci prípravy jednotlivých častí testovacieho prostredia. V samostatných podkapitolách je popísaný proces riešenia problémov v rámci priraďovania IP adries, prechodu z prihlasovania za pomoci hesla k autentizácii využitím verejného kľúču. Opis inštalácie operačného systému na zariadenia a charakteristika programov, ktoré vykonávajú testovanie šifrovacej výkonnosti.

V šiestej kapitole je vykonaná analýza výsledkov, ktoré boli v rámci testovania hardwarovej akcelerácie šifrovania zachytené. Obsahom tejto kapitoly je porovnávanie výsledkov medzi jednotlivými hardwarovými prípravkami, ktoré obsahujú hardwarovú podporu šifrovania, ako i porovnanie týchto vzoriek s jednotkami, ktoré hardwarové vylepšenie pre zrýchlenie šifrovania neobsahujú.

2. KRYPTOGRAFIA

Pod pojmom kryptografia rozumieme matematicko-vednú disciplínu, ktorá sa zaoberá skúmaním a vytváraním metód určených pre pretváranie nezašifrovanej správy do podoby šifrovaného textu. Hlavným dôvodom pre vznik a vývoj kryptografie a šifrovania správ obecne, je snaha zamedziť tretím osobám možnosť neoprávnene nahliadnuť a prípadne manipulovať s údajmi správy. Kryptografia spolu s krypto-analýzou[1] a steganografiou[2] tvorí dokopy celok vednej náuky kryptológie [1]. Kryptografiu, môžeme rozdeliť na symetrický šifrovací systém (Časť 2.3), asymetrický šifrovací systém (Časť 2.4) a hashovacie funkcie (Časť 2.5).

2.1 Šifrovanie

Šifrovanie je proces, ktorý prevádza obyčajný čitateľný text (Plaintext) do zašifrovanej podoby (Ciphertext). Poznáme viaceré druhy šifrovacích metód, ktoré sa od seba odlišujú podľa toho, aký šifrovací algoritmus je použitý. Samotné šifrovanie existuje už od staroveku v podobe napríklad cézarovej šifry alebo gréckeho skytale[3]. Tieto šifrovacie metódy sú dnes už zastaralé a považované za tzv. klasické šifrovacie algoritmy. Hlavným cieľom týchto metód bolo utajenie informácie. Moderné šifrovanie je realizované pomocou matematických operácií a počítačových algoritmov. Proces šifrovania môžeme matematicky definovať vzťahom (2.1).

$$C = E_K(M)\,, \tag{2.1}$$

Kde C je zašifrovaný text
E_K je proces šifrovania s použitím kľúču
K je kľuč použitý v šifrovaní
M je nezašifrovaná správa

[1] Veda zaoberajúca sa analýzou kryptografických systémov za účelom získania vedomostí potrebných na prelomenie šifrovania.
[2] Pod týmto pojmom rozumieme metódy pre ukrytie existencie správy samotnej. Na rozdiel od toho, kryptografia sa zaoberá len ukrývaním významu správy, ale nie skrývaním jej existencie.
[3] Drevený valec s predom dohodnutým priemerom, na ktorý sa navinoval pergamen. Šifrovanie spočívalo v tom, že sa čitateľná správa na pergamene ukázala len vtedy, pokiaľ mal drevený valec príjemcu a odosielateľa rovnaký priemer. Inak nedávala správa zmysel.

2.2 Dešifrovanie

Pod pojmom dešifrovanie rozumieme proces, ktorý prevádza zašifrovanú podobu správy (Ciphertext) do opätovného tvaru čitateľného textu (Plaintext). Tento proces môže byť matematicky definovaný vzťahom (2.2) na:

$$M = D_K(C)\ , \tag{2.2}$$

Kde M je nezašifrovaná správa

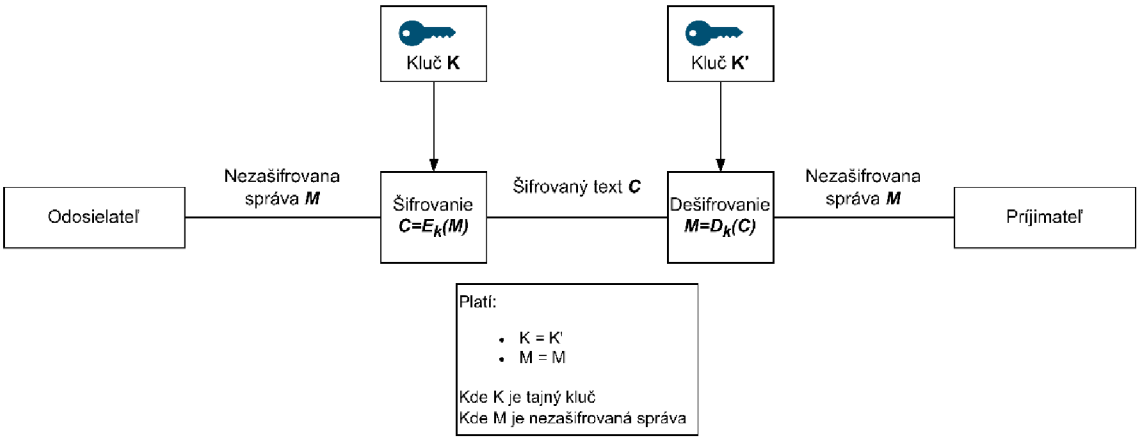
D_K je proces dešifrovania s použitím kľúču

K je kľuč použitý v dešifrovaní

C je zašifrovaný text.

2.3 Symetrický šifrovací systém

Niekedy taktiež nazývaný ako „šifrovanie pomocou jedného kľúča“, je charakteristický tým, že používa jeden rovnaký kľuč, ktorý je distribuovaný medzi všetkými účastníkmi komunikácie, tento kľuč slúži na šifrovanie a dešifrovanie správ [2] (viď. Obrázok 2.1). Do roku 1976, kedy bol navrhnutý asymetrický šifrovací systém, bol tento druh systému považovaný za jediný možný [3].

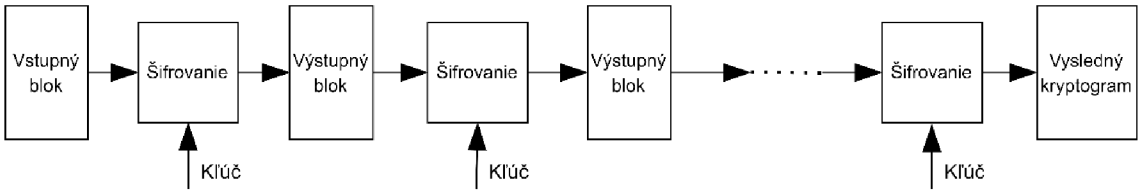


Obrázok 2.1 Princíp symetrického šifrovania

Hlavnou nevýhodou toho šifrovacieho systému je práve potreba distribuovať rovnaký kľuč všetkým účastníkom komunikácie, čo môže mať za následok ľahšie získanie kľúča použitého na šifrovanie a dešifrovanie správ, a tým pádom aj ľahšie získanie informácii. Šifrovanie pomocou tohto systému sa skladá z viacerých matematických procesov, medzi najhlavnejšie patrí substitúcia, permutácia a kombinácia. Symetrické šifry možno rozdeliť do dvoch kategórii, a to na prúdové šifry a blokové šifry. Najznámejšie symetrické šifry sú DES, TDES a AES (Rijndael).

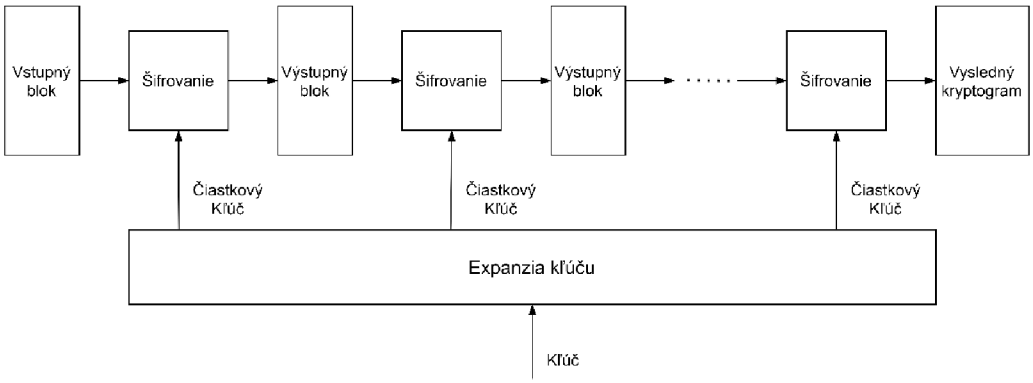
2.3.1 Bloková šifra

Šifrovanie sa prevádza po blokoch, kde každý blok má pevnú bitovú dĺžku n-bitov, pričom konštrukcia moderných blokových šifier je založená na princípe kaskádových šifier. Pod pojmom kaskádové šifry rozumieme zreťazenie viacerých šifier, kde výstupný blok jednej šifry je vstupný blok do šifry nasledujúcej (viď. Obrázok 2.2). Výstupný blok poslednej šifry je zároveň aj výsledným kryptogramom.



Obrázok 2.2 Princíp kaskádových šifier [4]

Často je používaný variant, kedy je zreťazených viacero rovnakých šifier. Tento typ šifry je nazývaný iterovaná šifra. Zo šifrovacieho kľúča sa v bloku expanzie odvodí niekoľko čiastkových kľúčov, ktoré sú neskôr použite ako šifrovacie kľúče pre iterovanú transformáciu. Pod pojmom iterovaná transformácia rozumieme zvyčajne pomerne jednoduchú šifru, v ktorej je vstupný blok pomocou iteračného kľúča a kombinačných blokových operácii (substitúcia, rotácia a aritmetické operácie) pretvorený do podoby výstupného bloku. Výstupný blok je vďaka princípu kaskádových šifier použitý ako vstupný blok pre ďalšie kolo transformácie. Tento proces sa opakuje až po posledný blok šifry, kedy sa z výstupného bloku iterovanej transformácie stáva výstupný blok zašifrovaného textu. Princíp iterovanej šifry možno vidieť na Obrázku 2.3 [4] [5].



Obrázok 2.3 Princíp iterovanej šifry [4] [5]

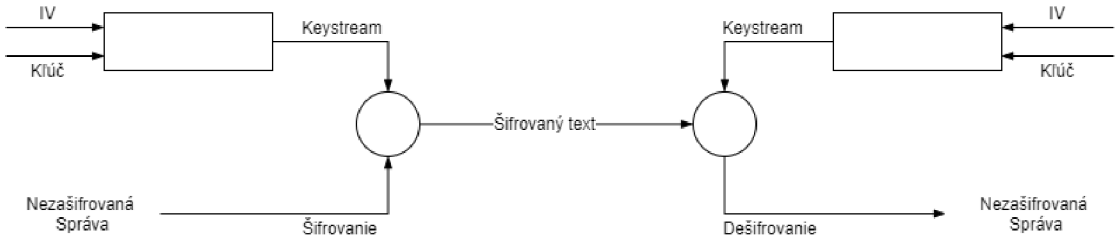
Medzi najznámejší typ iterovanej šifry patrí napríklad Feistelova šifra, v ktorej sa vstupný blok rozdeľuje na dve polovice. Hlavnou vlastnosťou tejto šifry je to, že v rámci jednej

iterácie dochádza k zašifrovaniu len jednej polovice šifry. Preto je dôležité, aby bol počet iterácii tejto šifry deliteľný dvomi a zároveň aby bol ich počet aj dostatočne veľký k tomu aby vo výslednej šifre dochádzalo k difúzii[4].

2.3.2 Prúdová šifra

Šifrovanie je realizované v rámci jednotlivých bitov. K šifrovaniu a dešifrovaniu sa používa šifrovacia postupnosť (keystream). Prúdové šifry môžeme rozdeliť na synchrónne a asynchrónne.

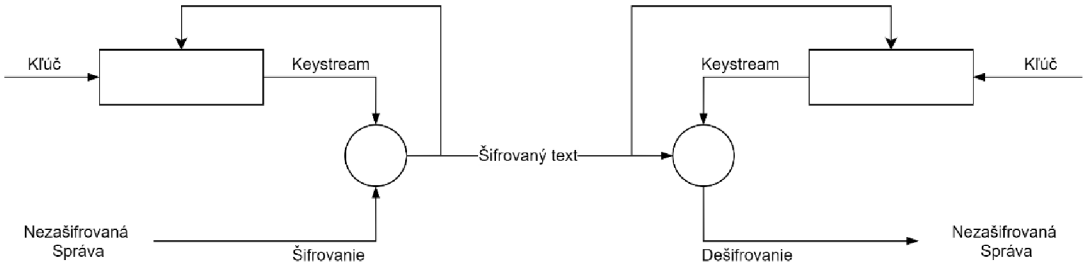
- Synchrónne prúdové šifry sú také, pri ktorých šifrovacia postupnosť závisí len na použitom kľúči, pričom však je aj potrebné, aby bola postupnosť medzi pôvodcom a adresátom zhodná a synchronizovaná [4]. V prípade, keby došlo k strate synchronizácie, tak nemôže v rámci konštrukcie šifry dôjsť k dešifrovaniu správy. Postupnosť je generovaná pomocou pseudonáhodného generátora čísel, kde pôvodca nastaví kľúčom generátor do rovnakého počiatočného stavu. V modernej prúdovej šifre je do nastavenia počiatočného stavu pseudonáhodného generátora ešte pridaný aj verejný inicializačný vektor [6]. Výhodou je, že v prípade chýb v prenose nie je ovplyvnená celá správa, ale iba chybne prenesené znaky. Princíp tohoto typu prúdovej šifry je zobrazený na Obrázku 2.4.



Obrázok 2.4 Princíp synchrónnej prúdovej šifry [7]

- Asynchrónne prúdové šifry sú typy šifier, kde keystream závisí okrem kľúču aj na počte písmen *t* už vytvorenej šifrovanej správy. Hlavným rozdielom oproti synchrónnym prúdovým šifrám je však to, že pokiaľ by malo prísť k desynchronizácii, tak asynchrónna prúdová šifra má schopnosť opätovnej synchronizácie po niekoľkých po sebe idúcich správnych znakoch šifrovanej správy. Pokiaľ dôjde k zmene jedného bitu počas prenosu šifrovaného textu, tak dešifrovanie prebehne, avšak chybne v niekoľkých nasledujúcich *t* znakoch. Po tomto už nastane bezchybný prenos [8]. Princíp tohoto typu prúdovej šifry je zobrazený na Obrázku 2.5.

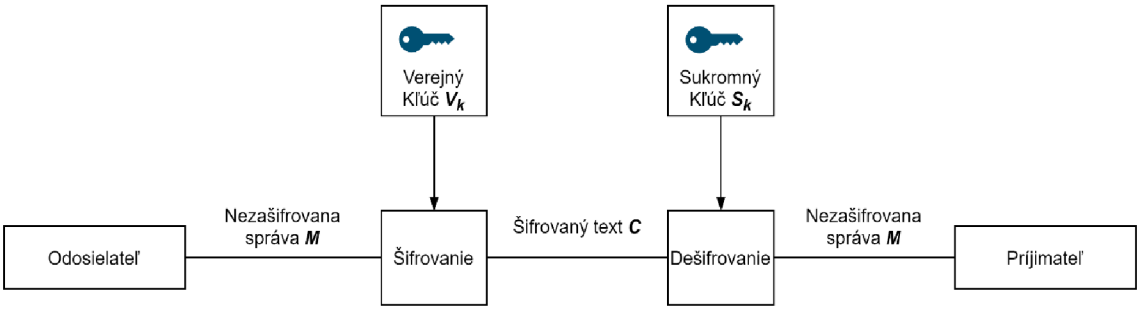
[4] Hodnota každého bitu výstupného bloku komplexne závisí na všetkých bitoch vstupného bloku.



Obrázok 2.5 Princíp asynchrónnej prúdovej šifry [8]

2.4 Asymetrický šifrovací systém

Snaha o zvýšenie zabezpečenia systémov umožnila v 70. rokoch, vzniku asymetrickému šifrovaniu. Na rozdiel od symetrického šifrovania, v tomto systéme existujú dva druhy kľúčov, jeden nazývaný súkromný kľuč (private key), ktorý je generovaný lokálne. Slúži na dešifrovanie správy a zároveň sa medzi účastníkmi komunikácie nevymieňa. Druhým typom kľúča je verejný kľuč (public key), ktorý je známy všetkým účastníkom komunikácie, je určený na šifrovanie správy (viď. Obrázok 2.6). Hlavnou charakteristikou týchto systémov je nemožnosť zistiť súkromný kľuč z originálneho šifrovacieho algoritmu, pretože sú použité jednosmerné matematické funkcie[5] [9].



Obrázok 2.6 Princíp asymetrického šifrovania

Asymetrické šifrovanie môže byť podľa princípu použitia v kryptografických systémoch rozdelené na:

- Šifrovanie/Dešifrovanie – Odosielateľ šifruje správu pomocou prijímateľovho verejného kľúču a prijímateľ správu naopak dešifruje pomocou svojho súkromného kľúča. Táto funkcia sa prevažne využíva vtedy, keď sa jedná

[5] Pod týmto pojmom rozumieme matematické funkcie, pre ktoré je možné ľahko z ľubovoľného vzoru vypočítať obraz, avšak pre tento obraz by malo byť prakticky nemožné spätne získať jeho vzor. V zahraničnej literatúre sú nazývané ako metódy padajúcich dverí [9]. V reálnych kryptografických systémoch sa používajú funkcie u ktorých sa dôveruje, že vlastnosť jednosmernosti majú.

o menšie súbory. Toto je z toho dôvodu, že asymetrické šifrovacie systémy sú veľmi náročné na implementáciu a výkon.

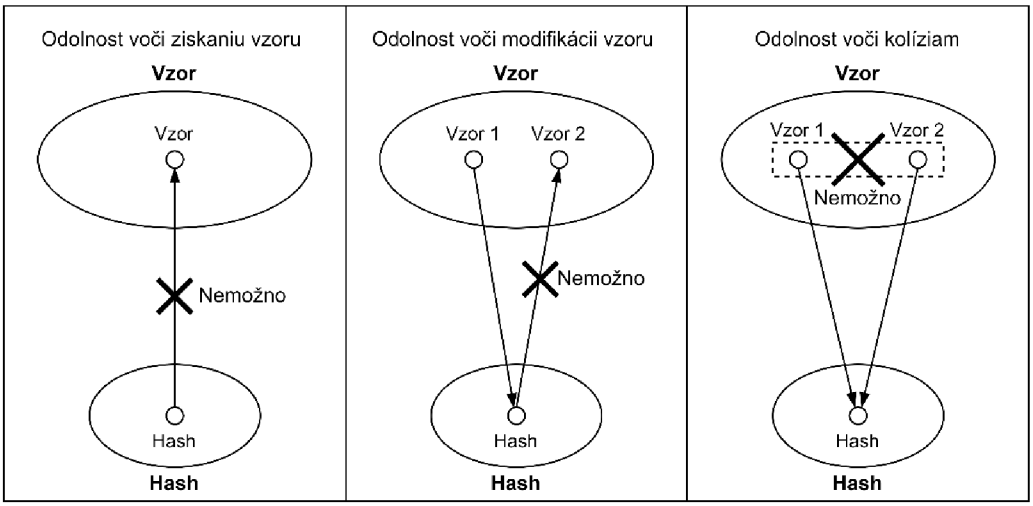
- Digitálny podpis – Odosielateľ vytvorí pomocou hashovacej funkcie jednosmerný hash, ktorý potom svojím súkromným kľúčom zašifruje a tým vytvorí podpis správy, ktorý spolu so správou odošle prijímateľovi. Prijímateľ použije odosielateľov verejný kľuč na rozšifrovanie vytvoreného podpisu a následne použije rovnaký hashovací algoritmus k vytvoreniu nového hashu. Tieto dve hodnoty príjemca porovná, a tým zistí, či ide skutočne o odosielateľa správy [10].
- Výmena Kľúčov – Obidvaja účastníci sa zúčastňujú na výmene tajnej hodnoty pomocou asymetrického šifrovacieho systému. Táto hodnota je ďalej použitá v rámci symetrickej kryptografie v podobe tajného kľúča (secret key) [9].

Medzi najznámejšie šifrovacie systémy patrí napríklad RSA alebo Diffie-Hellman. Nevýhodou tohto typu systému je tá, že na rozdiel od symetrického šifrovania, pracuje s výrazne výpočtovo pomalšími operáciami a kladie väčšie nároky na výkon.

2.5 Hashovacie funkcie

Pod týmto pojmom rozumieme kompresnú jednosmernú funkciu, ktorá priraďuje správe o ľubovoľnej bitovej dĺžke určitý obraz s pevne stanovenou dĺžkou, prevažne 128 až 512 bitov. Hashovacia funkcia musí spĺňať viaceré vlastnosti k tomu, aby bola považovaná za bezpečnú, tieto vlastnosti sú definovane podľa [4] [5] na:

- Odolnosť voči získaniu vzoru – pod týmto pojmom rozumieme vlastnosť, aby pre daný hash h bolo možno čo najviac nepravdepodobné to, že útočník získa taký vzor V, pre ktorý by platilo že $H(V) = h$. Táto vlastnosť sa prejavuje napr. v prípade útoku na hash hesiel, kedy zamedzuje spätnému získaniu hesiel [11].
- Odolnosť voči modifikovaniu zdroju – touto vlastnosťou sa rozumie to, aby k danému vzoru V_1 s priradeným hashom h_1 bolo takmer nemožné nájsť nejaký ďalší vzor V_2, pre ktorý by platilo že by mal rovnaký výstup hashovacej funkcie ako pôvodný vzor $H(V_2) = h_1$. Táto vlastnosť je dôležitá napríklad v rámci digitálnych podpisov [11].
- Odolnosť voči kolíziám – týmto rozumieme vlastnosť, kedy by malo byť takmer nemožné nájsť nejakú dvojicu rôznych vzorov $V_1 \neq V_2$ s rovnakou hodnotou hashu $H(V_1) = H(V_2)$, pri tejto vlastnosti vzniká problém s dĺžkou správy. V rámci toho, že správa je prevažne niekoľko násobne väčšia než dĺžka použitého hashu. Hashovacie funkcie použité v kryptografií majú vlastnosť, že ku kolíziám dochádza len veľmi zriedkavo. Pričom, čím viacej času potrebujeme na nájdenie kolízie, tým môžeme považovať hashovaciu funkciu za bezpečnejšiu [11].



Obrázok 2.7 Požadované vlastnosti hash funkcie [5]

Medzi najznámejšie hashovacie funkcie patrí napríklad MD5 a SHA1 a či SHA2. Pod pojmom SHA2 si môžeme predstaviť rodinu hashovacích funkcii SHA 224 až SHA 512.

2.6 Prevádzkové režimy blokových šifier

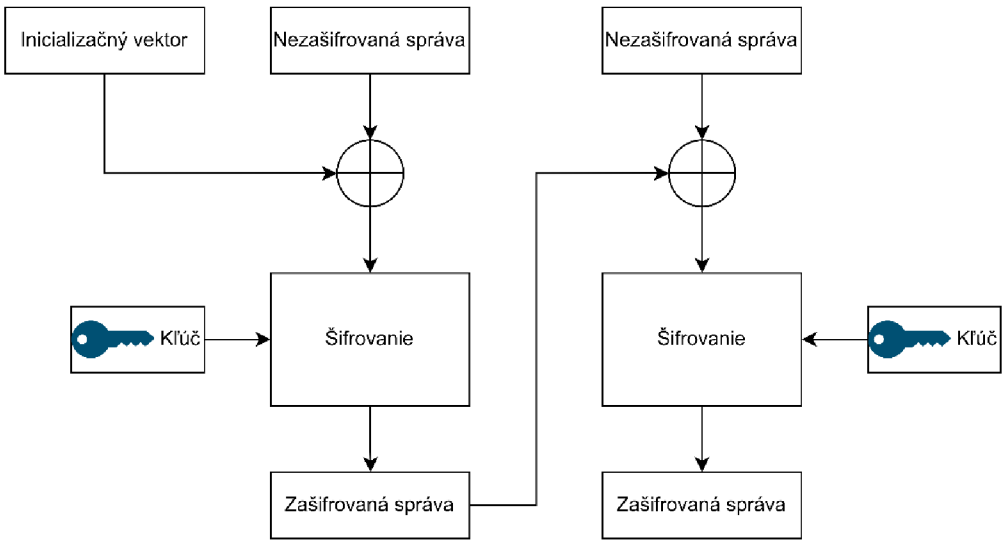
Bloková šifra sama o sebe dokáže zašifrovať a dešifrovať blok originálnej správy o pevnej dĺžke *n* bitov. V prípade, kedy je správa väčšia ako veľkosť bloku blokovej šifry, dochádza k využívaniu prevádzkových režimov. Tieto prevádzkové režimy definujú algoritmy, ktoré sú použité na šifrovanie a dešifrovanie správy dlhšej ako jeden blok [12]. Existuje 5 základných prevádzkových režimov. ECB (režim kódovej knihy), CBC (režim reťazenia šifrových blokov), CFB (režim šifrovej spätnej väzby), OFB (režim výstupnej spätnej väzby), CTR (čítačový režim). Režimy ECB a CBC vyžadujú, aby bola veľkosť nezašifrovanej správy celočíselným násobkom veľkosti *n* bitového bloku blokovej šifry. V prípade, kedy toto nie je splnené, je potreba nezašifrovanej správe pridať výplň niekoľkých bitov, potrebných pre získanie celočíselného násobku *n* bitov bloku blokovej šifry. V prípade, že je táto požiadavka splnená, tak sa na koniec musí pripojiť blok o veľkosti *n* bitov blokovej šifry ktorý obsahuje iba výplňové bity [13].

2.6.1 ECB (kódová kniha)

Režim kódovej knihy je považovaný za najjednoduchší prevádzkový režim blokovej šifry. Po rozšírení správy o výplň, je text rozdelený na niekoľko *n* bitových blokov. V ďalšom kroku je každý blok šifrovaný individuálne. Dešifrovanie prebieha taktiež po jednotlivých blokoch. Hlavná nevýhoda tohto režimu je v tom, že neskrýva vzory správ. Pod týmto si môžeme napríklad predstaviť opakovanie v nezašifrovanej správe, čo má za následok, že útočník je schopný rozpoznať opakujúce sa časti zašifrovanej správy. Táto vlastnosť zohráva dôležitý faktor, prečo je tento prevádzkový režim v dnešnej dobe neodporúčané používať [13] [14].

2.6.2 CBC (reťazenie šifrových blokov)

V tomto režime sú uplatnené dve hlavné myšlienky, prvou je snaha o zreťazenie šifrovania všetkých blokov tak, aby šifrovaný text y_i závisel nielen na jednom bloku nezašifrovanej správy x_i, ale taktiež na všetkých predchádzajúcich blokoch. Druhou myšlienkou je snaha o znáhodnenie šifrovania pomocou inicializačného vektora. Náhodný inicializačný vektor nám umožňuje zašifrovať rovnaké správy pomocou rovnakého kľúča, pričom sa výstupný šifrovaný text sa bude od seba odlišovať. Režim pracuje na princípe, že nezašifrovaná sprava x je doplnená o výplň potrebnú na získanie celočíselného násobku veľkosti n bitov bloku a následne rozdelená do jednotlivých blokov x_i. Ďalším krokom, je operácia XOR medzi zašifrovaným blokom y_i správy x_i a nasledujúcim nezašifrovaným blokom správy x_{i+1}. Výsledok XOR operácie je ďalej zašifrovaný pomocou blokovej šifry, pričom táto operácia dá vzniku zašifrovanému bloku správy y_{i+1}. V prípade počiatočného bloku nezašifrovanej správy x_1 sa do nezašifrovanej správy pridáva inicializačný vektor [13]. Dešifrovanie prebieha opačným smerom, pričom tento režim umožňuje paralelizovať dešifrovanie [14]. Princíp toho režimu šifrovania je zobrazený na obrázku 2.8.

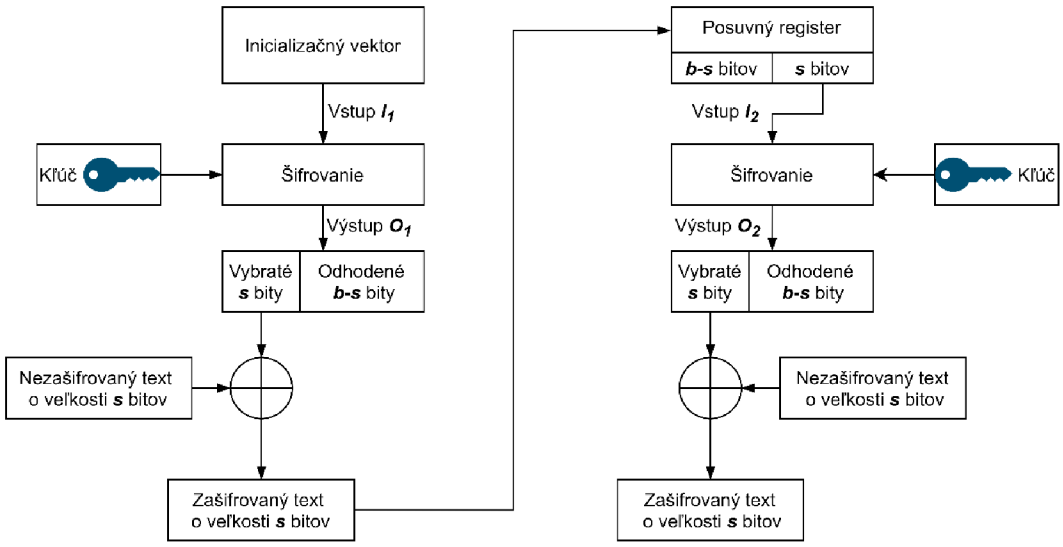


Obrázok 2.8 Princíp šifrovania pomocou CBC [14]

2.6.3 CFB (šifrová spätná väzba)

Pod týmto režimom si môžeme predstaviť pretvorenie blokovej šifry do podoby asynchrónnej prúdovej šifry [14]. Vďaka čomu, prídeme o nutnosť rozšíriť pôvodnú nezašifrovanú správu o vyplň potrebnú na získanie celočíselného násobku b bitov bloku blokovej šifry. Režim CFB potrebuje pre svoje fungovanie parameter s, pričom musí platiť že $1 \leq s \leq b$, kde s je kladné celé číslo a b je veľkosť bloku blokovej šifry v bitoch. Špecifické pre tento režim je rozdelenie blokov nezašifrovanej správy po

segmentoch s bitov. CFB pracuje na princípe, že vstupná hodnota šifrovacieho algoritmu je b-bitový posuvný register, ktorý je pri počiatočnom bloku nastavený na inicializačný vektor. Tento vektor je samostatne šifrovaný. Proces má za následok získanie výstupu šifrovacieho algoritmu. Najdôležitejšie bity (MSB) tohto výstupu sú pomocou operácie XOR zlúčené so segmentom nezašifrovanej správy, ostatné bity výstupného šifrovacieho algoritmu sú zahodené. Výsledkom tohto je získanie bloku šifrovanej správy o veľkosti s bitov [9]. Najmenej dôležite bity (LSB) $b - s$ inicializačného vektora sú spojené s s bitmi bloku šifrovanej správy k vytvoreniu ďalšieho vstupného bloku. Tento režim rovnako ako CBC umožňuje paralelizovať dešifrovanie [14] [15]. Princíp toho režimu šifrovania je zobrazený na obrázku 2.9.

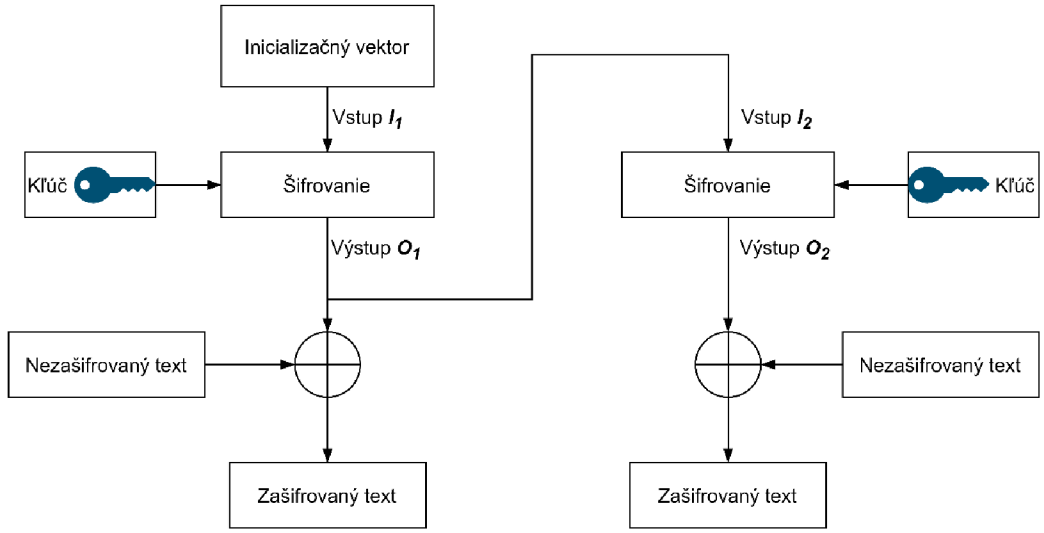


Obrázok 2.9 Princíp šifrovania pomocou CFB [9]

2.6.4 OFB (výstupná spätná väzba)

OFB je operačný režim, pod ktorým si môžeme predstaviť pretvorenie blokovej šifry do tvaru synchrónnej prúdovej šifry [14]. Toto nás zbavuje, rovnako ako v prípade CFB, nutnosti pridávať k pôvodnému nezašifrovanému textu vyplň, potrebnú na získanie celočíselného násobku n bitov bloku blokovej šifry. Na vstupe prvého bloku je použitý inicializačný vektor. Hlavným rozdielom oproti módu CFB je ten, že sa namiesto zašifrovanej správy privádza v pozícii vstupného bloku pre ďalšiu iteráciu blokovej šifry výstup šifrovacieho algoritmu. Ďalším rozdielom je ten, že OFB mód pracuje na celom bloku šifrovanej i nezašifrovanej správy, oproti čomu mód CFB pracuje na podmnožine s bitov. V prípade, že by posledný blok, n-bitovej blokovej šifry, nezašifrovaného textu obsahoval u bitov a zároveň by bolo splnené že $u < n$, tak dôjde k operácii XOR medzi posledným blokom správy a najdôležitejšími bitmi (MSB) posledného bloku výstupného šifrovacieho algoritmu. Výsledkom operácie je získanie posledného bloku zašifrovanej

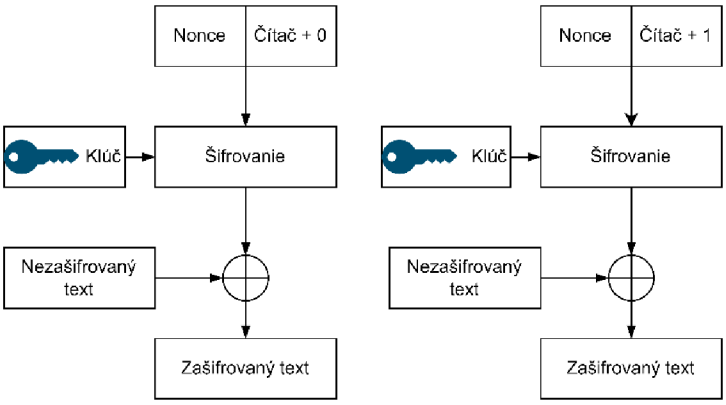
správy. Zvyšné bity $n - u$ šifrovacieho algoritmu budú zahodené. Ďalšou vlastnosťou tohto operačného módu je to, že šifrovanie a dešifrovanie prebieha na rovnakom princípe, pričom nie je možné paralelizovať ani dešifrovanie ani šifrovanie [9]. Princíp toho režimu šifrovania je zobrazený na obrázku 2.10.



Obrázok 2.10 Princíp šifrovania pomocou OFB [9]

2.6.5 CTR (režim čítača)

Pôvodne navrhnutý v roku 1979, tento operačný mód si môžeme taktiež predstaviť ako ďalšiu metódu pretvorenia blokovej šifry na tvar synchrónnej prúdovej šifry [14]. V tomto režime je požadované, aby každá hodnota čítača (counter) bola pre každý šifrovaný blok nezašifrovanej správy iná. Toto sa obyčajne dosahuje tak, že sa na čítači prvého počiatočného bloku nastaví hodnota, ktorá môže byť použitá v rámci šifrovania len raz (v angličtine sa tato hodnota nazýva nonce [9] [14]), pričom v praxi sa zvyčajne používa n bitový inicializačný vektor. Zároveň sa ku každej hodnote čítača pridáva číselná hodnota začínajúca od 0, ktorá je v každom kole šifrovania zväčšená o 1. Výsledok tejto operácie je ešte potrebné modulovať hodnotou 2^n, kde n je veľkosť vstupného bloku správy v bitoch, za účelom získania výstupného šifrovacieho algoritmu. Tento výstup je následne pomocou operácie XOR spojený s blokom nezašifrovanej správy, čo má za následok vytvorenie bloku zašifrovanej správy. Hlavným rozdielom oproti módu OFB, CFB a CBC je ten, že nedochádza k zreťazeniu šifrovania. Pre posledný blok nezašifrovanej správy bude vykonaný rovnaký proces ako pre OFB. Na zašifrovanie budú použité len najviac dôležité bity MSB šifrovacieho algoritmu, zvyšné bity budú zahodené. Tento mód umožňuje paralelizovať ako dešifrovanie, tak aj šifrovanie [9]. Šifrovanie a dešifrovanie prebieha rovnakým procesom. Fungovanie toho módu je zobrazené na obrázku 2.11.



Obrázok 2.11 Princíp šifrovania pomocou CRT [9]

2.7 AES

Symetrická bloková šifra s originálnym názvom Rijndael, ktorá bola navrhnutá v roku 2001 Americkým národným inštitútom pre štandardy a technológie, ako nástupca šifry DES. Do dnešnej doby považovaná za celosvetový šifrovací štandard. Šifra je schopná spracovávať dátové bloky o veľkosti 128 bitov a zároveň umožňuje výber medzi tromi hodnotami dĺžky kľúča (výber z možnosti 128, 196 či 256 bitov). Túto vlastnosť môžeme v rámci názvu šifry zobraziť ako AES-128, AES-196 či AES-256 [16].

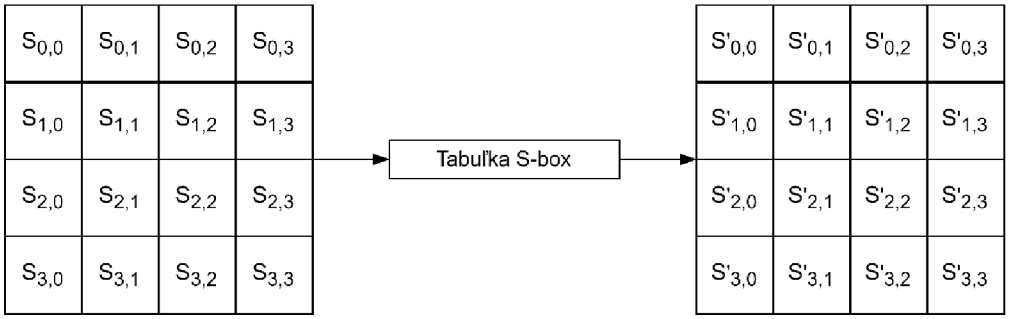
2.7.1 Vysvetlenie fungovania šifry na móde AES-128

Šifra je konštruovaná na konečnom telese GF[6] (2^8), pod týmto si môžeme predstaviť objekt s 256 prvkami. Ide o osem bitové čísla, ktoré nazývame bajt. Operácia sčítania dvoch prvkov je definovaná ako operácia XOR jednotlivých bitov. Násobenie je určené ireducibilným[7] polynómom r(x) = ($x^8+x^4+x^3+x+1$) [4].

AES pracuje s dvojrozmernými blokmi (maticami), pričom hlavná dátová jednotka šifry, na ktorej sú prevádzané všetky operácie, je matica o veľkosti 4 x 4 bajty nazývaná Stav. Pre tuto maticu sú definované štyri transformácie [4] [16]:

SubBytes – ide o nelineárnu bytovu substitučnú funkciu, ktorá sa nezávisle vykonáva na každom bajte Stavu. Ku každej hodnote vstupného bajtu je pomocou substitučnej tabuľky S-box, ktorá je definovaná v AES ako matica o veľkosti 16 x 16 za účelom pokrytia permutácii všetkých možných 256 8-bitových hodnôt, priradená hodnota výstupného bajtu. Hodnoty sú zachytené v hexadecimálnej jednotkovej sústave (viď. Tabuľka 2.1). Prvé štyri bity vstupného bajtu tvoria premennú x, ktorá definuje riadok tabuľky a posledné štyri bity tvoria premennú y, ktorá definuje stĺpec tabuľky [4]. Tieto hodnoty je treba previesť z binárnej na hexadecimálnu jednotkovú sústavu. V bunke o súradniciach (x, y) je uvedená hodnota výstupného bajtu substitúcie. Princíp fungovania je zobrazený na obrázku 2.12.

[6] Matematická definícia konečného telesa [17]
[7] Polynóm ktorý nemožno rozložiť na menšie polynómy

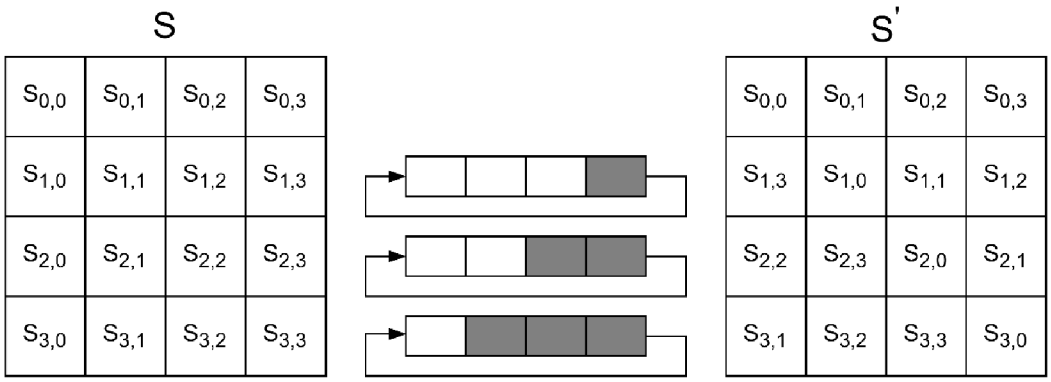


Obrázok 2.12 Princíp transformácie SubBytes [16]

Tabuľka 2.1 Tabuľka S-box [16].

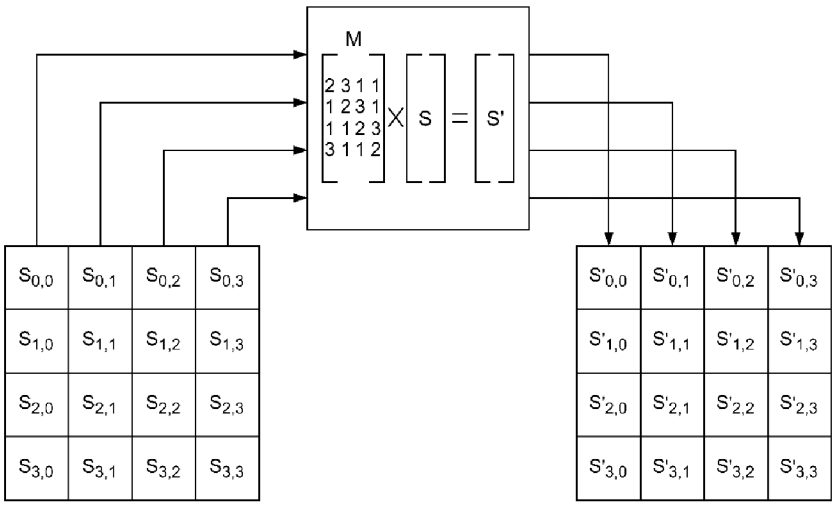
		y															
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

ShiftRows – Permutácia bajtov matice Stav v ktorej sa posledné tri riadky cyklicky posunu o rozdielny počet bajtov (offset) [16]. Prvý riadok matice sa neposúva. V druhom riadku sa vykoná posunutie vľavo o 1 bajt. V treťom riadku nastane o posunutie v rovnaký smer, avšak tento krát o 2 bajty. V štvrtom riadku je predvedené posunutie o 3 bajty [4]. Ukážka tejto transformácie je na obrázku 2.13.



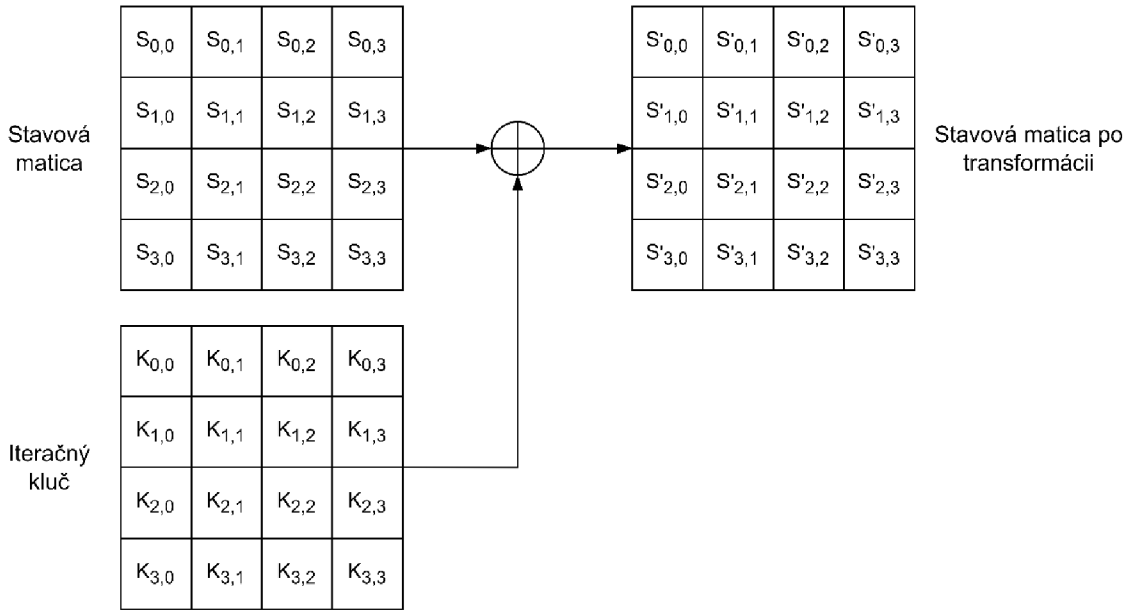
Obrázok 2.13 Princíp transformácie ShiftRows [9]

MixColumns – Účelom tejto transformácie je substitúcia stĺpcov matice Stav, táto činnosť je vykonaná na všetkých stĺpcoch samostatne [16]. Princíp tejto operácie spočíva v tom, že sa stĺpec *S* násobí s predom definovanou maticou *M*, vďaka čomu vznikne stĺpec *S'* transformovanej matice (viď. Obrázok 2.14). Hodnota každého bajtu stĺpca transformovanej matice je výsledkom sčítania všetkých elementov jedného riadka a jedného stĺpca, čo má za následok to, že výsledná hodnota je závislá na všetkých hodnotách bajtov stĺpca pôvodnej matice [4].



Obrázok 2.14 Princíp transformácie MixColumns [9]

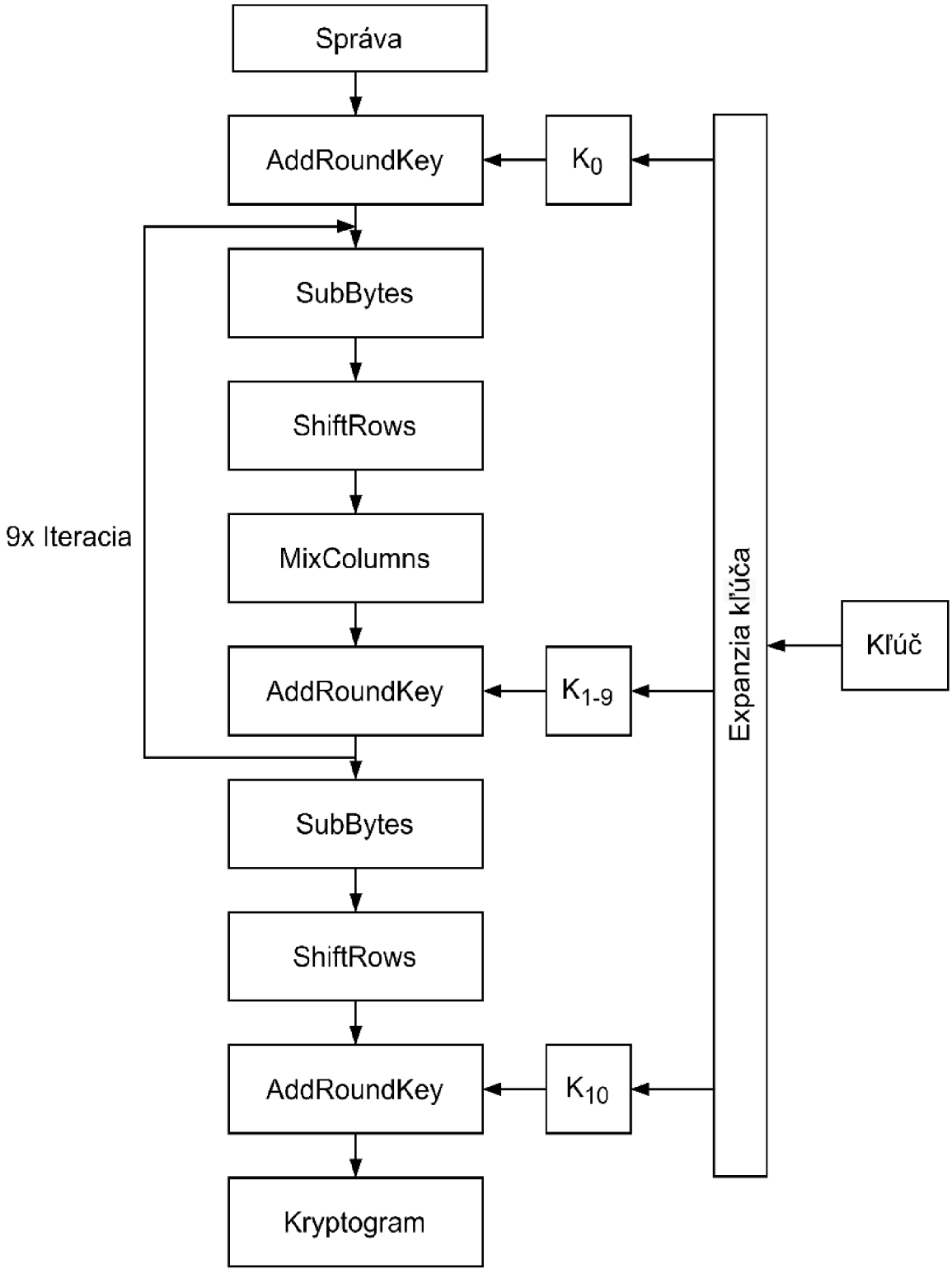
AddRoundKey – Pred začiatkom šifrovania sa predvádza expanzia kľúču, kde sa z kľúča *K* odvodzujú čiastkové kľúče, pričom každý ma rovnakú formu ako matica Stav. Ku každému bajtu stavovej matice sa operáciou XOR pridáva odpovedajúci bajt iteračného kľúča [4]. Princíp je ukázaný na obrázku 2.15.



Obrázok 2.15 Princíp transformácie AddRoundKey [4] [9]

2.7.2 Priebeh šifrovania pomocou AES

Samotné šifrovanie AES prebieha v rámci niekoľko kôl, pričom počet opakovaní sa odvíja od zvolenej veľkosti kľúča. S kľúčom o veľkosti 128 bitov dochádza k šifrovaniu v rámci 10 kôl, s kľúčom 192 zasa v rámci 12 a pri 256 bitoch v rámci 14 kôl [16]. K matici Stav, v ktorej sa nachádzajú bajty nezašifrovanej správy sa pri inicializácii pričíta čiastkový kľuč K_0. V nasledujúcich 9 kolách sa vykoná šifrovanie, ktoré sa skladá z operácii SubBytes, ShiftRows, MixColumns a AddRoundKey. Každé kolo iterácie používa svoj vlastný čiastkový kľuč K_1 – K_9. Vo finálnom desiatom kole iterácie sa nepoužíva operácia MixCollums, pričom výstupom tohto kola je matica Stav v ktorej sú uložene bajty kryptogramu [4] [9] [16]. Priebeh šifrovania je zobrazený na obrázku 2.16.

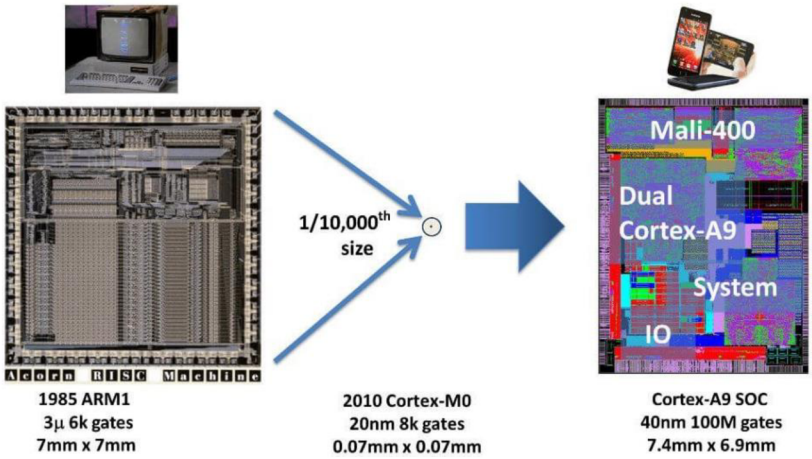


Obrázok 2.16 Princíp šifrovania pomocou AES [4] [9]

3. ARM

3.1 História

Firma ARM, ktorá dnes architektúru vyvíja, pôvodne vznikla v 80. rokoch, ako spoločný projekt spoločností Apple, Acorn Computers a VSLI, za účelom implementácie ARM technológie do zariadení Apple Newton. ARM vznikol z dôvodu, že Apple nechcel oficiálne používať súčiastky firmy Acorn Computers, ktorá bola vtedy považovaná za ich konkurenciu [18]. Prvá inštrukčná sada v programovacom jazyku BASIC od spoločnosti ARM, bola vyvinutá už v roku 1985 [19], avšak nebola použitá v žiadnom komerčnom produkte. Ďalší vývoj umožnil vzniknúť vylepšenej verzii ARMv2, v ktorej bola pridaná možnosť operácie násobenia a taktiež podpora pre koprocesory. Vlastnosťou týchto prvých sád bola tá, že používali len 26 bitový adresný priestor. Táto nevýhoda bola opravená v ďalšej verzii s názvom ARMv3, ktorá rozšírila použiteľné adresovanie na 32 bitov a obsahovala aj spätnú podporou pre predchádzajúce verzie. Ďalším významným pokrokom bolo pridanie možnosti zachovania aktuálneho stavu programu v prípade vzniku výnimky. ARMv4 umožnila ukladanie inštrukcii o rozdielnej dátovej dĺžke, avšak aj zároveň prestala spätne podporovať operácie pre 26 bitový adresný priestor. ARMv5 sa vyznačuje pridaním softwarových breakpointov[8] [20]. Týmto postupným vývinom sa dostávame až k procesorom rady ARM-Cortex. Do verzie ARMv8 podporovala tato architektúra len 32 bitové adresovanie a neskôr bola rozšírená o 64 bitové adresovanie vo forme AArch64 [21].



Obrázok 3.1 Ukážka miniaturizácie ARM procesorov, prebraté z [21]

[8] V slovenčine názov bod prerušenia. Slúži pre testovanie počas vývoju programov.

3.2 Architektúra

Procesory s inštrukčnou sadou od firmy ARM (Advanced RISC Machines) patria do rodiny tzv. RICS (Reduced Instruction Set Computing) procesorov. Pod pojmom RICS rozumieme architektúru CPU s malou, ale za to s veľmi špecializovanou inštrukčnou sadou[9]. Účelom zmenšenia sady je snaha, o zrýchlenie vykonávania jednotlivých inštrukcii. Hlavnou vlastnosťou RICS je implementácia architektúry load-store, v ktorej sú samostatne rozdelené príkazy na prácu s dátami (operácie, ako napríklad sčítanie, odčítanie či násobenie) a príkazy určené pre manipuláciu s pamäťou (operácie load a store) [22]. Architektúra ARM nadväzuje na tieto základy RICS tým, že pridáva sade viaceré vylepšenia. Medzi tieto môžeme zaradiť napríklad funkcionalitu pre načítavanie a ukladanie viacerých inštrukcii naraz, alebo aj implementáciu podmienok pre spustenie určitých inštrukcii. Posledná zmena bola pridaná za účelom optimalizácie vykonaného počtu pokynov, ktoré dokáže procesor spracovať [23]. Jadro ARM procesorov obsahuje veľký počet všeobecne zameraných registrov, pričom inštrukcie sa vykonávajú v rámci jedného cyklu. Ďalšou vlastnosťou je, že máju jednoduché režimy adresovania, kde všetky adresy načítania/ukladania možno určiť z obsahu registra a polí s pokynmi [24]. V architektúre ARMv8 bola ďalej pridaná podpora pre 64 bitové adresovanie so spätnou kompatibilitou pre 32 bitové adresovanie [25].

3.2.1 Rozdiel medzi RISC (ARM) a CISC (x86)

Hlavným rozdielom, týchto procesorov je v použitej inštrukčnej sade. Ako už bolo spomínané procesory ARM Cortex sú procesory postavené na RISC architektúre, zatiaľ čo procesory s x86 architektúrou od spoločnosti Intel využívajú strojové inštrukcie CISC architektúry. V porovnaní dvoch architektúr, najväčším rozdielom býva komplexnosť príkazov, ktoré dokážu jednotlivé procesory vykonať a energia spotrebovaná počas ich spracovania. Procesor typu x86 má schopnosť vykonávať jednoduché procesy, ako napríklad načítavanie z pamäte, ale zároveň môže aj vykonávať viacero komplexných príkazov, a toto všetko v rámci jednej inštrukcie. Táto vlastnosť, má za následok zväčšenie komplexnosti daného procesoru, v ktorom je vyhradený veľký kus vnútornej logiky na dekódovanie inštrukcii, a ich prevedením na vnútorné operácie. Na rozdiel od toho, ARM procesory majú menší, viacej všeobecne zameraný počet inštrukcii. Tieto inštrukcie môžu byť vykonané s použitím výrazne menej hardwarových prostriedkov, čo umožňuje znížiť cenu čipov a zároveň znížiť aj spotrebovanú energiu.

[9] Pod týmto si môžeme predstaviť strojový kód ktorý definuje inštrukcie, dátové typy, registre či hardwarovú podporu pre prácu s pamäťou.

3.3 Súčasnosť

V dnešnej dobe sa firma ARM tld. hlavne sústreďuje na vývoj a licencovanie procesorov z rady Cortex. Existujú viaceré profily, ktoré rozdeľujú pre aké zariadenie je procesor určený. Medzi najhlavnejšie patria [26]:

- A – Profil určený pre výkonovo náročné systémy, ako napríklad mobilné telefóny alebo tablety.
- R – Profil zameraný na výkon s čo najmenšou odozvou, využívaný v systémoch od ktorých závisí správne fungovanie zariadenia. Pod týmito systémami si môžeme predstaviť napríklad kontrolné jednotky v autách, alebo aj sieťové zariadenia. Hlavnou vlastnosťou tejto rady je to, že sa využíva pri procesoch v reálnom čase.
- M – Hlavné zameranie tohto profilu, je čo najnižšia cena a čo možno najnižšia spotreba energie. Neobsahuje jednotku na správu pamäte. Hlavné využitie nachádza v implementácii v senzoroch, alebo ako súčasť SoC obvodov.

3.4 Využitie

Arm procesory sa využívajú všade okolo nás, od najmenších riadiacich systémov až po tablety, notebooky, počítače a autá. Ponúkajú viacero možností, od veľmi efektívnych procesorov, ktoré majú nízky výkon, a tým nemusia byť chladené, až po procesory v superpočítačoch používané napríklad pre výskum Covid-19 [27]. Ďalším využitím arm procesorov je implementácia v rámci technológie vstavaných systémov [28], pod týmto pojmom si môžeme napríklad predstaviť zariadenia Raspberry Pi, alebo aj ako súčasť výkonovo obmedzených zariadení, ktoré sa používajú v oblasti IoT (Internet of Things). Do tejto kategórie spadajú napríklad detektory čistoty vzduchu, úrovne slnečného žiarenia, či aj nástroje pre nastavenie pozície antény [29].

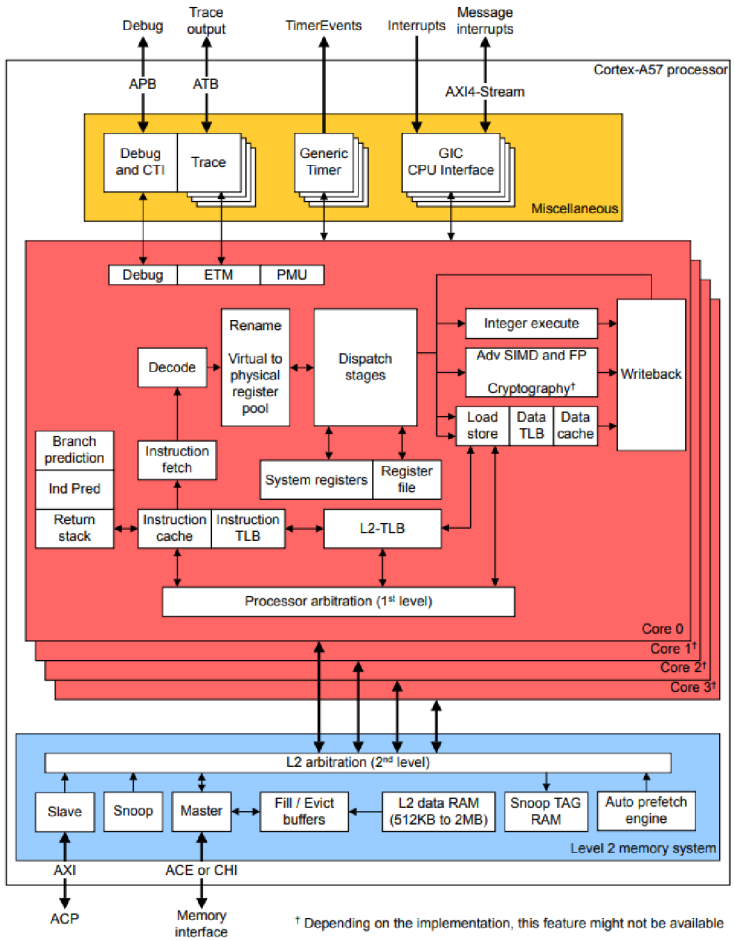
Obrázok 3.2 Superpočítač pre výskum Covid-19 [27]

3.5 Architektúra ARM Cortex-A

Architektúru Cortex-A môžeme rozdeliť do dvoch skupín. Rozdelená je podľa toho, či sa jedná o procesor s podporou 32 bitového adresovania, alebo s podporou 64 bitového adresovania. Procesory architektúry Cortex-A, sú určené na použite pri výkonovo náročných systémoch a vďaka implementácie jednotky na správu pamäte (Memory management unit), majú schopnosť používať operačné systémy. Tato vlastnosť je využívaná napríklad v jednodoskových počítačoch, ktoré používajú SoC čipy, či v smart mobilných telefónoch.

Tabuľka 3.1 Rozdelenie Procesorov podľa Architektúry.

Procesory architektúry ARMv7-A (32 bit)	Procesory architektúry ARMv8-A (64 bit)
Cortex-A5	Cortex-A35
Cortex-A7	Cortex-A53
Cortex-A9	Cortex-A57
Cortex-A15	Cortex-A72
Cortex-A17	Cortex-A73



Obrázok 3.3 Blokový diagram funkcie procesoru Arm Cortex-A57, prebraté z [30]

3.6 Podpora pre kryptografiu

V ARM procesoroch s architektúrou ARMv8, je možnosť v blokových diagramoch vidieť aj voliteľné rozšírenie pre registre určené na kryptografiu. Toto však nie je súčasťou základnej verzie procesorov Cortex-A, z dôvodu potreby ďalších licencii (viď. Obrázok 3.3). Obsahom rozšírenia je hardwarová akcelerácia pre šifrovanie pomocou AES, a zároveň aj pridanie podpory pre hashovacie algoritmy SHA-1, SHA-224 a SHA-256. Väčšina výrobcov v svojich doskách implementuje vlastný čip zaoberajúci sa bezpečnosťou a hardwarovou akceleráciou pre kryptografiu (viď. Časť 4.2).

3.6.1 Zistenie prítomnosti podpory pre kryptografiu v hardwarovom prípravku

Vo väčšine prípadov je možné v Unix-like operačných systémoch použiť príkaz: *cat proc/cpuinfo* pre získanie podrobnejších informácii o procesore použitom v danom zariadení. V rámci tohto dokážeme zistiť vlastnosti jednotlivých procesorov, ako je napríklad hardwarová podpora pre šifrovanie, či podpora hashovacích algoritmov.

```
root@orangepi3:~# cat /proc/cpuinfo
processor       : 0
BogoMIPS        : 48.00
Features        : fp asimd evtstrm aes pmull sha1 sha2 crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd03
CPU revision    : 4
```

Obrázok 3.4 Procesor s podporou šifrovania, obrázok je invertovaný pre väčšiu citeľnosť

```
root@rpi4b:~# cat /proc/cpuinfo
processor       : 0
BogoMIPS        : 108.00
Features        : fp asimd evtstrm crc32 cpuid
CPU implementer : 0x41
CPU architecture: 8
CPU variant     : 0x0
CPU part        : 0xd08
CPU revision    : 3
```

Obrázok 3.5 Procesor bez podpory šifrovania, obrázok je invertovaný pre väčšiu citeľnosť

4. TESTOVANIE VÝKONU

4.1 Metrika testovania

Cieľom testovania, je zmeranie výkonu použitých procesorov v rámci procesu šifrovania. Pre kvantifikovanie testovaných hodnôt, je potrebné nadefinovať čo je to výkon v rámci kryptografie (viď. Časť 4.1.1). Hodnoty budú merané s desať násobným opakovaním. Za účelom eliminácie štatistickej odchýlky či prípadnej chyby v meraní, bude zo skupín hodnôt vytvorený aritmetický priemer.

4.1.1 Kryptografický výkon

Pod týmto pojmom rozumieme, koľko krát môžeme zašifrovať jeden blok správy o veľkosti *n* bajtov, za určité časové obdobie *t*.

4.2 Testovaný hardware

Testovanie minipočítačov, prebieha na platformách ktoré obsahujú procesory z rady Cortex-A. ARM nevyrába vlastné čipy, ale len predáva licenčné pravá pre svoju procesorovú architektúru viacerým výrobcom, ktorí tuto architektúru implementujú vo svojich integrovaných obvodoch SoC (System on Chip). Tieto sa od seba môžu odlišovať aj v prípade rovnakého procesoru. Táto podkapitola, sa zameriava na vymenovanie použitého hardwaru, ktorý bude následne použitý v meraní.

Tabuľka 4.1 Zoznam použitého hardwaru v testovaní

Názov dosky	Názov SoC (System on Chip)	Použitý ARM procesor	Počet jadier	Rok výroby
Raspberry Pi 4B	Broadcom BCM2711 [31]	Cortex A-72	4	2019
OrangePi Lite	AllWinner H3 [32]	Cortex A-7	4	2016
OrangePi Zero Plus 2 H3	AllWinner H3 [33]	Cortex A-7	4	2017
NanoPi K1 Plus	AllWinner H5 [34]	Cortex A-57	4	2018
NanoPi M4	Rockchip RK3399 [35]	Cortex A-72	2	2018
OrangePi 3	AllWinner H6 [36]	Cortex A-53	4	2019
Rock Pi S	Rockchip RK3308 [37]	Cortex A-35	4	2019

4.2.1 Broadcom BCM2711

Čip vyrobený od spoločnosti Broadcom špeciálne pre dosku Raspberry Pi 4 model B. Čip v sebe obsahuje štvorjadrový 64bitový procesor Cortex A72 a 32bitový grafický procesor Videocore VI [31]. Bohužiaľ tento čip neobsahuje rozšírenie pre kryptografiu.

4.2.2 Allwinner H3

Architektúra od spoločnosti Allwinner Technology, určená pre prijímanie televízneho prenosu cez internet. Čip SoC, ktorý v sebe obsahuje štvorjadrový procesor Cortex A7 spoločne s koprocesorom NEON a zabudovaným grafickým procesorom ARM Mali400MP2 s grafickou akceleráciou. Vlastnosťou tohto typu architektúry je to, že má v sebe zabudovanú hardwarovú akceleráciu pre šifrovacie algoritmy, ako sú napríklad AES, DES alebo TDES [38].

4.2.3 Allwinner H5

Ďalším vývojom architektúry H3 dalo vzniku architektúre H5. Hlavným rozdielom oproti predošlej verzii je to, že štvorjadrový procesor Cortex A-7 bol nahradený za štvorjadrový procesor Cortex A-57, ktorý umožňuje spracovať ako 64bitové tak aj 32bitové inštrukcie. Grafický procesor bol vylepšený na Mali 450 MP4. Rovnako ako v architektúre H3 je zabudovaná hardwarová akcelerácia pre šifry AES, DES a TDES [39].

4.2.4 Rockchip RK3308

SoC architektúra, od výrobcu Fuzhou Rockchip Electronics, určená pre spracovanie zvukového signálu a ďalších digitálnych multimédií. Čip ma zabudovanú technológiu na detegovanie ľudského hlasu a prípadne aj reakcie na ľudské príkazy. Tento čip nemá žiadny grafický procesor, ale len štvorjadrový procesor Cortex A-35. Čip v sebe obsahuje aj kryptografické rozšírenie pre ARMv8, čo znamená, že rovnako ako predchádzajúco spomenuté architektúry, má zabudovanú hardwarovú akceleráciu pre šifrovanie pomocou AES, DES a TDES [40].

4.2.5 Rockchip RK3399

Tento integrovaný obvod je zaujímavý tým, že v sebe obsahuje 2 rozdielne procesory. Tieto, A-72 a A-52, sa spolu nachádzajú v rámci jednej architektúry ARM big.LITTLE. Pod týmto rozumieme spojenie výkonnejšieho, ale zato viacej energeticky náročného procesoru so slabším ale energeticky úspornejším. Účelom je vytvorenie viacjadrového CPU, ktoré sa môže viacej prispôsobiť potrebe užívateľa v rámci výkonu [41]. Čip obsahuje kryptografické rozšírenie architektúry ARMv8 spolu s podporou pre šifrovanie pomocou AES, TDES. Tento obvod podporuje, ako pseudonáhodný, tak aj pravý generátor náhodných čísiel [42].

4.2.6 Allwinner H6

Architektúra obsahujúca štvorjadrový procesor A-53, určená pre domáce televízie systémy za účelom prehrávania videí z internetu. Obsahuje čip pre kryptografickú akceleráciu, ktorý podporuje viaceré druhy šifrovacích algoritmov s podporou paralelného šifrovania, medzi ktoré patrí napríklad AES a všetky jeho prevádzkové režimy. Ďalej do skupiny patria hashovacie funkcie SHA alebo MD5. V rámci architektúry je implementovaný aj ďalší bezpečnostný systém, ktorý je používaný pri

šifrovaní SD kariet, tento systém umožňuje hardwarovú akceleráciu šifrovania pomocou AES pri použití režimu ECB alebo CBC [43].

4.3 Testovacie prostredie

Okrem použitého hardwaru, je potrebné nadefinovať operačný systém a software, ktorý bude použitý pri testovaní jednotlivých ARM procesorov a dosiek.

4.3.1 Operačný systém

Z pomedzi veľkého množstva operačných systémov pre architektúru ARM, bolo treba vybrať práve ten, ktorý bude najviac vyhovovať naším potrebám pre testovanie. Na výber boli napríklad, operačný systém Windows on ARM, Arch Linux, Ubuntu, Debian či prípadne port Ubuntu a Debianu s názvom Armbian. Hlavným kritériom pre rozhodovanie, bola v prvom rade podpora hardwaru použitého v testovaní. Pod týmto pojmom rozumieme skutočnosť, či vôbec dokážeme daný operačný systém spustiť na použitom hardware. Nasledujúcim kritériom bola licenčná a cenová politika operačného systému. Operačný systém Windows on ARM vyžaduje pre možnosť svojho požívania zaplatenie za licenciu a užívateľ si ho nemôže ľubovoľne upravovať, zatiaľ čo Linuxové distribúcie sú voľne stiahnuteľné na stránkach ich autorov a taktiež ich je možné voľne modifikovať. Ďalším kritériom bola náročnosť na inštaláciu, Arch linux je určený pre pokročilejších užívateľov, čo má za následok veľmi náročnú inštaláciu. Rozhodujúcim kritériom, ktoré nakoniec umožnilo výber bola celková podpora systému.

Výsledný operačný systém, ktorý spĺňal všetky tieto podmienky, bol port Ubuntu a Debianu špecializovaný na ARM pre väčšie množstvo dosiek s názvom Armbian.

4.3.2 Použitý software

OpenSSL

Program bol využitý v rámci testovania, kvôli nástrojom z knižnice libcrypto, ktorá implementuje viaceré kryptografické algoritmy. Na vyber nájdeme napríklad hashovacie funkcie SHA-256, či blokovú šifru AES s viacerými možnosťami pre prevádzkové režimy. Pre nás je dôležitý príkaz speed, ktorý umožňuje testovať jednotlivé kryptografické algoritmy [44].

```
root@nanopim4:~# openssl
OpenSSL> version
OpenSSL 1.1.1f  31 Mar 2020
OpenSSL> enc -list
Supported ciphers:
-aes-128-cbc               -aes-128-cfb               -aes-128-cfb1
-aes-128-cfb8              -aes-128-ctr               -aes-128-ecb
-aes-128-ofb               -aes-192-cbc               -aes-192-cfb
-aes-192-cfb1              -aes-192-cfb8              -aes-192-ctr
-aes-192-ecb               -aes-192-ofb               -aes-256-cbc
-aes-256-cfb               -aes-256-cfb1              -aes-256-cfb8
-aes-256-ctr               -aes-256-ecb               -aes-256-ofb
-aes128                    -aes128-wrap               -aes192
-aes192-wrap               -aes256                    -aes256-wrap
-aria-128-cbc              -aria-128-cfb              -aria-128-cfb1
-aria-128-cfb8             -aria-128-ctr              -aria-128-ecb
-aria-128-ofb              -aria-192-cbc              -aria-192-cfb
-aria-192-cfb1             -aria-192-cfb8             -aria-192-ctr
-aria-192-ecb              -aria-192-ofb              -aria-256-cbc
-aria-256-cfb              -aria-256-cfb1             -aria-256-cfb8
-aria-256-ctr              -aria-256-ecb              -aria-256-ofb
-aria128                   -aria192                   -aria256
-bf                        -bf-cbc                    -bf-cfb
-bf-ecb                    -bf-ofb                    -blowfish
-camellia-128-cbc          -camellia-128-cfb          -camellia-128-cfb1
-camellia-128-cfb8         -camellia-128-ctr          -camellia-128-ecb
-camellia-128-ofb          -camellia-192-cbc          -camellia-192-cfb
-camellia-192-cfb1         -camellia-192-cfb8         -camellia-192-ctr
-camellia-192-ecb          -camellia-192-ofb          -camellia-256-cbc
-camellia-256-cfb          -camellia-256-cfb1         -camellia-256-cfb8
-camellia-256-ctr          -camellia-256-ecb          -camellia-256-ofb
-camellia128               -camellia192               -camellia256
-cast                      -cast-cbc                  -cast5-cbc
-cast5-cfb                 -cast5-ecb                 -cast5-ofb
-chacha20                  -des                       -des-cbc
-des-cfb                   -des-cfb1                  -des-cfb8
-des-ecb                   -des-ede                   -des-ede-cbc
-des-ede-cfb               -des-ede-ecb               -des-ede-ofb
-des-ede3                  -des-ede3-cbc              -des-ede3-cfb
-des-ede3-cfb1             -des-ede3-cfb8             -des-ede3-ecb
-des-ede3-ofb              -des-ofb                   -des3
-des3-wrap                 -desx                      -desx-cbc
-id-aes128-wrap            -id-aes128-wrap-pad        -id-aes192-wrap
-id-aes192-wrap-pad        -id-aes256-wrap            -id-aes256-wrap-pad
-id-smime-alg-CMS3DESwrap  -rc2                       -rc2-128
-rc2-40                    -rc2-40-cbc                -rc2-64
-rc2-64-cbc                -rc2-cbc                   -rc2-cfb
-rc2-ecb                   -rc2-ofb                   -rc4
-rc4-40                    -seed                      -seed-cbc
-seed-cfb                  -seed-ecb                  -seed-ofb
-sm4                       -sm4-cbc                   -sm4-cfb
-sm4-ctr                   -sm4-ecb                   -sm4-ofb
```

Obrázok 4.1 Ukážka podporovaných šifrovací algoritmov programu OpenSSL, obrázok invertovaný pre väčšiu prehľadnosť

OpenSSH

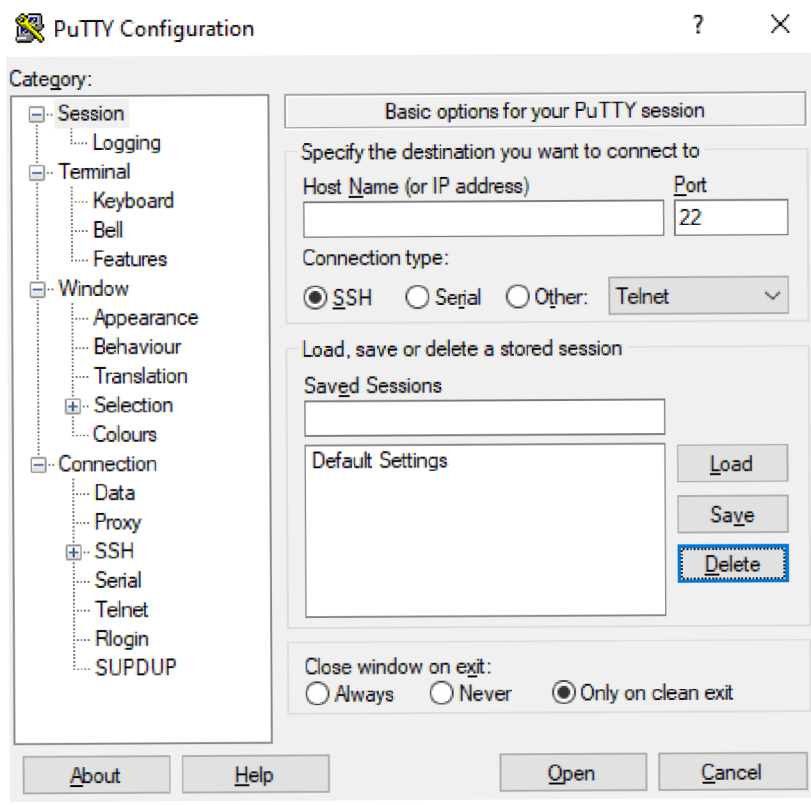
Program pôvodne vyvinutý v roku 1999, ako súčasť systému OpenBSD. Dôvod pre vznik, bola snaha o odstránenie obmedzujúcich licencii pre používanie protokolu SSH. Pod pojmom SSH, si vieme predstaviť umožnenie vzdialeného pripojenia k zariadeniu s prístupom k internetu cez nezabezpečenú sieť. Hlavnou výhodou, oproti protokolom ktoré, vykonávajú rovnakú funkciu (napríklad Telnet či Rlogin) je ten, že používa šifrovanie komunikácie. SSH má dve rozdielne verzie, ktoré nie sú spolu kvôli použitiu rozdielnych šifrovací algoritmov kompatibilné. Verzia 1.0 je dnes považovaná za nebezpečnú na používanie, z dôvodu chýb v šifrách použitých v zabezpečení. Program OpenSSH, je do dnes aktualizovaný ako samostatný open-source projekt, ktorý je napríklad používaný, ako súčasť operačného systému Ubuntu či Debian. OpenSSH dnes, už umožňuje, len pripojenie pomocou SSH 2, kde je používané asymetrické šifrovanie pomocou DSA a výmena kľúčov pomocou algoritmu Diffie-Hellman. Zaujímavosťou je, že program preferuje špecializovanú kryptografickú knižnicu LibreSSL, ktorá vznikla v roku 2014 z OpenSSL [45].

```
root@nanopim4:~# ssh -V
OpenSSH_8.2p1 Ubuntu-4ubuntu0.4, OpenSSL 1.1.1f  31 Mar 2020
root@nanopim4:~# ssh
usage: ssh [-46AaCfGgKkMNnqsTtVvXxYy] [-B bind_interface]
           [-b bind_address] [-c cipher_spec] [-D [bind_address:]port]
           [-E log_file] [-e escape_char] [-F configfile] [-I pkcs11]
           [-i identity_file] [-J [user@]host[:port]] [-L address]
           [-l login_name] [-m mac_spec] [-O ctl_cmd] [-o option] [-p port]
           [-Q query_option] [-R address] [-S ctl_path] [-W host:port]
           [-w local_tun[:remote_tun]] destination [command]
```

Obrázok 4.2 Ukážka programu OpenSSH, obrázok invertovaný pre väčšiu prehľadnosť

Putty

Klientský program, ktorý umožňuje prepojenie so vzdialeným systémom pomocou Telnetu, SSH či Rloginu. Ďalej obsahuje aj podporu pre lokálne pripojenie pomocou sériovej linky. Program v sebe neuchováva z dôvodu bezpečnosti heslá, avšak obsahuje možnosť pre užívateľa použiť svoj verejných kľúč ako prostriedok k overeniu pri prihlasovaní. Program je dostupný, ako na operačnom systéme Windows, tak aj na unixových distribúciách. Putty ma podporu ako SSH 2, tak aj SSH 1, pričom v poslednom menovanom prípade, musí dôjsť k vyžiadaniu od užívateľa [46].



Obrázok 4.3 Ukážka programu Putty

5. PRÍPRAVA TESTOVACIEHO PROSTREDIA

Prvotným úkonom, v rámci prípravy testovacieho prostredia, bolo nastavanie statických IP adries pre jednotlivé hardwarové prípravky. Pôvodne, bolo použitie priraďovanie adries pomocou DHCP, ktoré priraďovalo IP adresy z rozsahu 192.168.1.100 až 192.168.1.254, toto sa ukázalo, ako neoptimálne. Problémom, bola neprítomnosť logickej postupnosti a náhodné priraďovanie IP adries jednotlivým doskám. Toto malo za následok to, že v počiatočných fázach testovania, dochádzalo k preskočeniu náhodného počtu IP adries, ako i k problémom s identifikovaním aktuálne komunikujúceho zariadenia. Ako riešenie tohto problému, boli jednotlivým zariadeniam priradené statické adresy (viď. Tabuľka 5.1).

Tabuľka 5.1 Statické IP adresy hardwarových prípravkov

Názov dosky	IP Adresa
Raspberry Pi 4B	192.168.1.127
Orange Pi 3	192.168.1.126
Orange Pi Zero Plus 2 H3	192.168.1.125
Orange Pi Lite	192.168.1.124
Nano Pi K1 Plus	192.168.1.123
Nano Pi M4	192.168.1.122
Rock Pi S	192.168.1.121

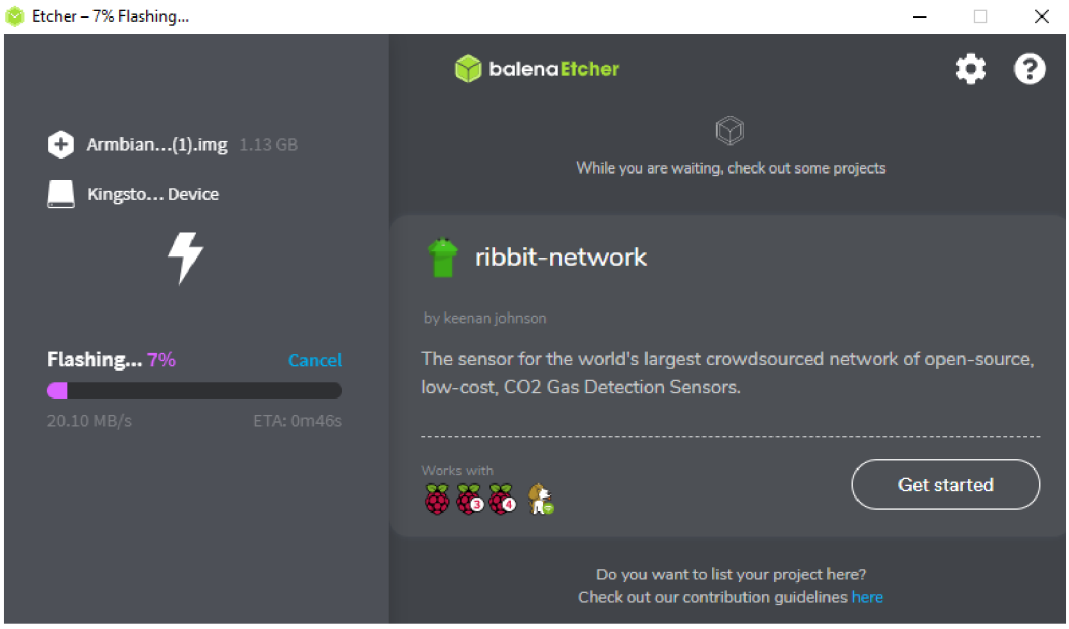
5.1 Autentizácia pomocou verejného kľúča

Ďalšou prekážkou, sa počas testovania ukázala byť, opakovaná potreba zadávať užívateľské údaje pri vzdialenom prihlasovaní, pomocou protokolu SSH. Táto vlastnosť sa vyskytovala i pri využívaní programov, ktoré sú na danom protokole postavené (v našom prípade sa jedná o program s názvom SCP, ktorý je určený na kopírovanie súborov medzi klientom a vzdialeným hostiteľom). Táto požiadavka, mala za následok to, že nebolo možné proces testovania automatizovať a v prípade zle zadaného hesla, dokonca testovanie kompletne ukončiť. Za účelom odstránenia tejto prekážky, bola využitá možnosť alternatívnej autentizácie, pomocou verejného kľúča. Postup pre nastavenie a aktiváciu je nasledovný:

```
1.)Vytvorenie verejného RSA kľúču na strane klienta
ssh-keygen -t rsa
2.)Vytvorenie priečinka .ssh na strane hostiteľa
ssh root@192.168.1.125 mkdir -p .ssh
3.)Vytvorenie súboru authorized key v priečinu .ssh na strane hostiteľa
do ktorého zároveň kopírujeme verejný kľuč
cat .ssh/id_rsa.pub | ssh root@192.168.1.125 'cat >> .ssh/authorized_keys'
4.)Nastavenie práv pre správne fungovanie autentizácie
ssh root@192.168.1.125 "chmod 700 .ssh"
ssh root@192.168.1.125 "chmod 600 .ssh/authorized_keys"
```

5.2 Inštalácia operačného systému na zariadenia

Platforma Armbian, poskytuje na svojich webových stránkach, možnosť stiahnutia optimalizovaných systémových obrazov pre jednotlivé hardwarové prípravky. Tieto obrazy sú založené, buď na operačných systéme Ubuntu alebo Debian a nemožno ich medzi jednotlivými hardwarovými prípravkami meniť, z dôvodu ich špecializácie na procesorové architektúry, ako i na inú hardwarovú výbavu jednotlivých prípravkov. Súbor formátu IMG sa po stiahnutí nachádza v bezstratovom kompresnom formáte xz. K napáleniu na SD karty, používame program balenaEtcher [47], v ktorom špecifikujeme cestu k stiahnutému súboru a vyberáme médium, ktoré bude použité na zápis. Program následne naformátuje médium a dekomprimuje xz subor. Po úspešnej dekomprímácii, dochádza k napáleniu IMG súboru na SD kartu a následne ešte k overeniu správneho zápisu.



Obrázok 5.1 Ukážka programu balenaEtcher

5.3 Príprava testovacích dát

Dáta pre testovanie rýchlosti šifrovania, boli pripravené v operačnom systéme Windows 10. Jedná sa o textové súbory s reálnym obsahom o veľkosti 1, 4, 8, 16, 32, 64, 128 a 256 megabajtov. Generovanie prebieha v príkazovom riadku, pomocou príkazu echo a for cyklu, ktorý pri každej iterácii zdvojnásobuje veľkosť súboru tým, že si vypíše obsah súboru pomocou príkazu type a tento výstup pripája na koniec pôvodného súboru.

```
echo|set /p="VUTFEKTVBRNEVUTF" > 1MB.txt
for /L %i in (1,1,16) do type 1MB.txt >> 1MB.txt
```

Príkaz set /p použitý zároveň s príkazom echo, zabraňuje pridaniu nového riadka. Parameter /L v for cyklu špecifikuje, že sa jedná o cyklus s rozsahom číslic.

5.4 Testovacie programy

Testovanie prípravkou je vykonávane na linuxovej platforme Armbian, pomocou skriptov vytvorených v skriptovacom jazyku Bash, ktorých výsledky vo forme textových súborov sú následne spracovávané pomocou programov v programovacom jazyku Python. Úlohou, týchto programov, je spracovávať výsledné textové súbory, na súbory formátu csv. Z týchto súborov, sú následne pomocou programovacích knižníc Pandas, Numpy a Matplotlib, vytvárané grafy pre jednotlivé prípravky a merané šifrovacie algoritmy. Hlavným dôvodom pre výber práve týchto dvoch programovacích jazykov je ten, že sa v operačnom systéme nachádzajú už po prvotnej inštalácii. V podkapitolách 5.4.1 a 5.4.2 je bližšie popísaná funkcionalita programov. Tieto skripty a programy môžeme nájsť v elektronickej prílohe.

5.4.1 Bash skript

Na testovanie šifrovania, bolo v operačnom systéme Armbian použitých päť skriptov, prvý súbor s názvom connectionscript.sh, slúži na automatické aktualizovanie testovacích skriptov, ako i testovacích dát.

```
#!/bin/bash
for ip in 192.168.1.{121..127}
do
      echo $ip
      ssh root@$ip 'if [[ -d ~/Dummy_Data ]]; then rm -r Dummy_Data;
      rm encryptionscript.sh; rm speedscript.sh ;echo "Cleanup Done";
      else echo "Nothing to clean"; fi'
      scp -r Dummy_Data/ root@$ip:~/
      scp encryptionscript.sh root@$ip:~/
      scp speedscript.sh root@$ip:~/
done
```

Druhým krokom vo for cyklu, je vzdialené pripojenie k hostiteľovi pomocou SSH, v rámci if podmienky, dochádza k zisteniu existencie zložky Dummy Data v hostiteľskom domovom adresári. Pokiaľ daná zložka existuje, tak dochádza k odstráneniu, ako jej tak i súborov obsahujúcich testovacie skripty. Zmazané položky, následne daný skript, pomocou programu SCP, znovu do hostiteľského systému nahrá.

Ďalším v poradí, je súbor speedscript.sh, tento testovací skript, bol použitý na opakované zachytávanie výstupu príkazu *openssl speed* do textového súboru. Zaujímavosťou je, že program OpenSSL, vypisuje výsledky príkazov ako do štandardného výstupu (Stdout), tak i do štandardného chybového výstupu (Stderr). Z dôvodu, zachytenia všetkých testovacích dát, bolo pomocou súborových deskriptorov presmerované Stdout a Stderr do samostatných textových súborov.

```
do
      openssl speed -mr -evp $str 2> ${str}_Overviewtemp.txt 1>
      ${str}_Summarytemp.txt
      sed "s/$/:${i}:"${HostName}"/" ${str}_Summarytemp.txt | tee -a
      ${str}_Summary
      sed "s/$/:${i}:"${HostName}"/" ${str}_Overviewtemp.txt | tee -a
      ${str}_Overview
done
```

Ďalej, je pomocou príkazu *tee* k vytvoreným textovým súborom pridávaný názov testovaného hardwaru a aktuálna iterácia merania. Tento proces, je následne desať krát zopakovaný pre každú šifru. Po dokončení cyklu, sa odstránia dočasné súbory s príponou temp a textový súbor sa presunie do odkladacieho priečinku. Tento priečinok je vytvorený vždy, pred začiatkom testovacieho cyklu a slúži na odloženie výstupných textových súborov jednotlivých operačných módov šifry AES s rovnakou veľkosťou kľúča (AES-128, AES-192 a AES-256). Po ukončení poslednej šifry, sa vytvorí zložka s názvom užívateľa do ktorej budú premiestnené všetky odkladacie priečinky.

Nasledujúci súbor, encryptionscript.sh je použitý na zachytenie času potrebného pre vykonanie reálneho šifrovania súborov s použitím príkazu *openssl enc*. Prvotným nápadom na implementáciu, bolo použitie zabudovaného shell príkazu s názvom *time*, avšak po vykonanom testovaní, bolo od tejto myšlienky z dôvodu maximálnej presnosti na tri desatinné miesta upustené. Náhradou, za toto sa stal výpočet pomocou zachytenia času v nanosekundách, príkazom *date* pred a po spustení príkazu na šifrovanie a ich následným odčítaním.

```
do
      ((iterationtext=iterationtext+1))
      PreScriptDate=$(date +"%s.%N")
      openssl enc -"${MasterArray}" -k 12345 -in "${filename128}" -out
      enc_"$iterationtext" -nosalt
      PostSpeedDate=$(date +"%s.%N")
      rm enc_$iterationtext
      Diff2=$(echo "$PostSpeedDate - $PreScriptDate" | bc)
      echo "${filename128##*/};${HostName};${MasterArray};${Diff2};${i}" >>
      "${MasterArray}"_txt_output.txt
done
```

Do premenných, PreScriptDate a PostSpeedDate, bol pomocou príkazu *date* ukladaný čas, pričom pre požiadavku v rámci presnosti, bolo potrebné v úvodzovkách špecifikovať formát výstupu. Jedná sa o kombináciu `%s.%N`, kde `%s` špecifikuje to, že je požadovaný výpis hodnoty ubehnutého času v sekundách od dátumu 1.1.1970[10], k tomu pomocou `%N` ešte pridávame aj rozšírenie aktuálneho času v nanosekundách. Tieto dve hodnoty, sú od seba oddelené desatinou bodkou.

Nasledujúcim problémom, v tomto prípade bola skutočnosť, že Bash sám o sebe nedokáže vykonávať aritmetické operácie s premenenými, ktoré obsahujú dátový typ float. Riešením, tohto nedostatku je predávanie výstupu externej aplikácii s názvom *bc*, ktorá zastáva funkcionalitu vedeckej kalkulačky.

```
shopt -s nullglob
PathToTextFile=(~/Dummy_Data/Text/*)
for MasterArray in "${myarray[@]}"
    do
        for i in {1..10..1}
            do
            ShuffledPath=( $(shuf -e "${PathToTextFile[@]}") )
```

[10] Alternatívny názov je Linux Epoch

Za účelom, čo najviac objektívneho testovania a zabránenia optimalizácie operačným systémom počas priebehu šifrovania, používame príkaz *shuf*, ktorý vytvorí z poradia ciest k testovaným súborov náhodnú permutáciu. Tento proces sa bude opakovať 10 krát v rámci for cyklu.

Pre zlepšenie automatizácie a odstránenie potreby prítomnosti SSH pripojenia pri testovaní sú použité dva skripty, ktoré využívajú programu GNU *Screen* pre odpojenie obrazovky pri vykonávaní jednotlivých testovaní.

```
for ip in 192.168.1.{121..127}
do
      ssh root@$ip 'screen -S script -d -m bash speedscript.sh'
done
```

5.4.2 Python program

V rámci testovania sú použité dva programy - *parse_encryption_speed* a *parse_openssl_speed_output*. Na formátovanie údajov textových súborov do formátu csv je použitá knižnica CSV. Na následné matematické a štatistické manipulácie s vytvorenými súbormi je použitá Python knižnica Pandas. Pre samotné grafovanie zachytených údajov je využitá knižnica MatPlotLib, ktorá obsahuje utility pre vykreslenie a manipuláciu s vykreslenými grafmi. V rámci automatizácie nachádzania súborov a vytvárania zložiek boli k importovaným knižniciam pridané ešte Glob a OS.

Účelom programu *parse_encryption_speed* je spracovávanie výsledných súborov z testovania rýchlosti šifrovania reálnych dát jednotlivých hardwarových prípravkov za pomoci skriptu *encryptionscript.sh*. Výstupy tohto programu sú nasledovné: vektorový graf vo formáte svg ktorý zachytáva jednotlivé operačné módy šifry AES, graf obsahujúci všetky módy šifry v rámci jednej veľkosti použitého kľúču šifry AES a následne i graf zobrazujúci jednu šifru v rámci viacerých hardwarových prípravkov.

Ďalším programom v poradí je *parse_openssl_speed_output,* ktorý spracováva zachytenú priepustnosť získanú zo skriptu *speedscript.sh.* Z týchto spracovávaných údajov je vyberané minimum a maximum v rámci všetkých iterácii danej šifry. K týmto zaznamenaným hodnotám, program následne pridáva ešte aj vypočítaný aritmetický priemer zo všetkých iterácii. Formáty výstupov sú rovnaké ako predchádzajúcom programe.

6. VYHODNOTENIE TESTOV

Táto kapitola, popisuje výsledky testovania procesorov. Testovanie, bolo uskutočnené pomocou skriptov, ktoré nám zachytávali výstupy príkazov určených pre meranie výkonnosti šifrovania. Výsledky testov, boli spracované do tabuliek, pričom hlavným účelom bolo porovnanie šifrovacej výkonnosti jednotlivých prípravkov. Následne, aj porovnanie zariadení, obsahujúcich hardwarovú akceleráciu s minipočítačmi bez tohto rozšírenia. Testovanie, prebiehalo v dvoch etapách, v prvej bolo pomocou skriptu *speedscript* zisťovaná priepustnosť jednotlivých šifier, viď. podkapitola 6.1. V druhej časti, bola skúmaná, rýchlosť šifrovacích algoritmov v rámci šifrovania reálnych testovacích dát s rozdielnou veľkosťou, viď. podkapitola 6.2. Za účelom, zlepšenej prehľadnosti, sú ku každej časti pridané grafy, ktoré reprezentujú zachytené hodnoty pre hardware s akceleráciou a bez akcelerácie.

6.1 Šifrovanie pomocou speedscript-u

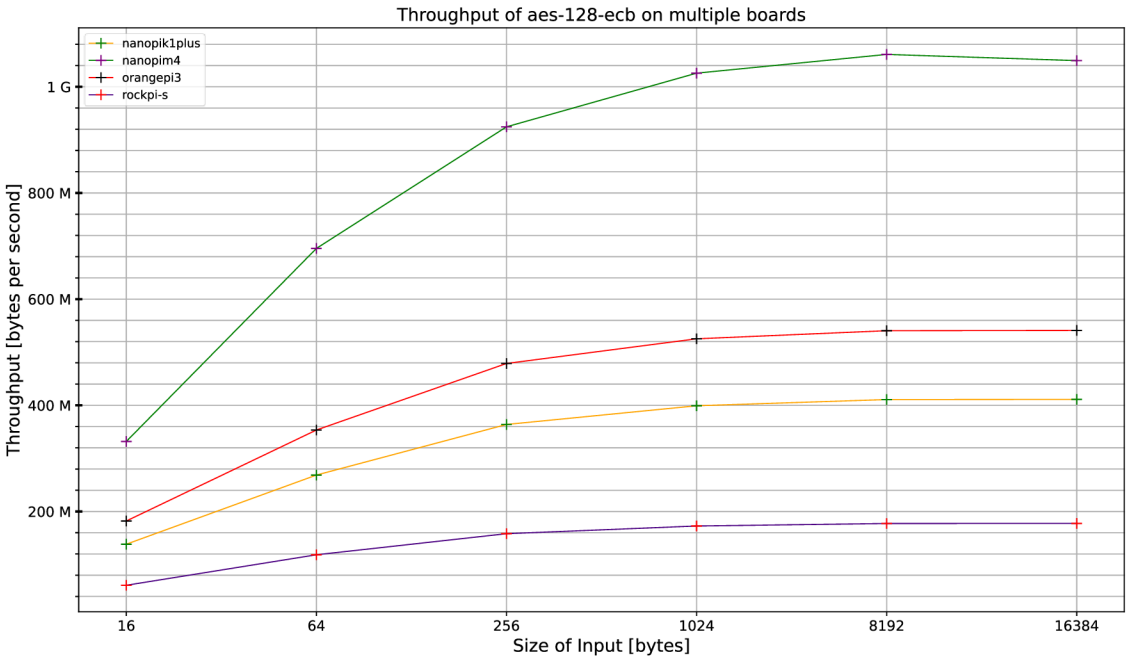
Testovanie šifrovania, prebieha v rámci skriptu *speedscript* (viď. Časť 5.4.1), ktorý používa program OpenSSL, pričom veľkosť vstupu je prednastavená na hodnoty v rozsahu 16, 64, 256, 1024, 8192 a 16384 bajtov, zároveň je dĺžka testovania prednastavená na 3 sekundy. V tabuľke 6.1 vidíme hodnoty pre vstup o veľkosti 16 bajtov. Ďalšie hodnoty sa nachádzajú v elektronickej prílohe.

Tabuľka 6.1 Testovanie priepustnosti pomocou skriptu *speedscript* pre operačné módy šifry AES-128 s veľkosťou vstupu 16 bajtov

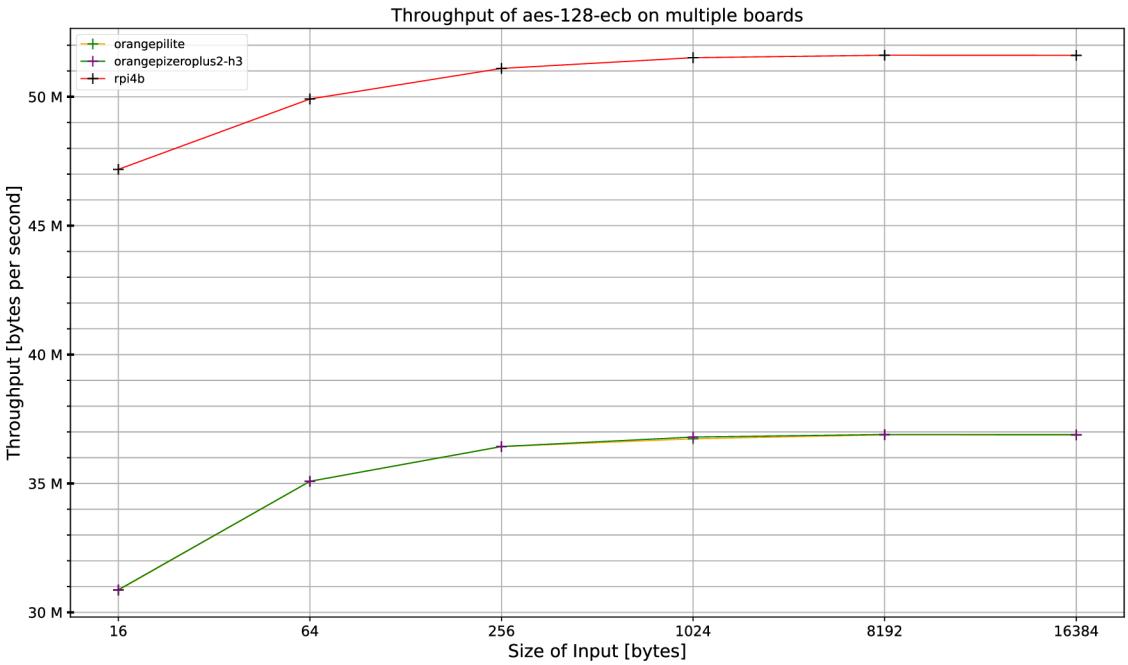
Šifra / Doska	AES-128-ECB	AES-128-CBC	AES-128-CFB	AES-128-OFB	AES-128-CTR
Raspberry Pi 4B	47184429,826	45521911,539	43242103,772	44189725,952	41575998,933
Orange Pi Lite	30858130,654	25748688,011	24969999,210	25457290,386	24406529,321
Orange Pi Zero +2 H3	30875710,822	25761481,341	24968175,467	25503436,861	24451618,115
Nano Pi K1 Plus	138346801,602	114819194,666	90200245,869	95544677,334	79954539,941
Nano Pi M4	331883796,647	267711816,065	193567605,780	203546446,075	171889696,022
Orange Pi 3	182083726,933	151083579,157	117121033,066	125491235,992	106700268,871
Rock Pi S	61027233,601	48159191,396	39507168,000	43177052,057	33953223,088

Prípravky, ktoré sú v tabuľke zobrazené tučne a podčiarknuté, reprezentujú prípravky obsahujúce hardwarovú akceleráciu. Z údajov vyplýva, že prípravky bez akcelerácie šifrovania, majú výrazne menšiu priepustnosť v porovnaní s prípravkami, ktoré hardwarovú akceleráciu obsahujú (Tieto hodnoty sa od seba odlišujú minimálne dvojnásobným zväčšením). Zaujímavosťou však je, že doska Rock Pi S, ktorá i navzdory

toho, že má v sebe implementovanú hardwarovú akceleráciu, tak dosahuje porovnateľnú priepustnosť ako Raspberry Pi 4B. Toto je z dôvodu, že sa v Rock Pi S nachádza menej výkonný procesor.



Obrázok 6.1 Testovanie výkonnosti pomocou príkazu speed, dosky s hardwarovou akceleráciou



Obrázok 6.2 Testovanie výkonnosti pomocou príkazu speed, dosky bez hardwarovej akcelerácie

Ako, môžeme vidieť z grafov na obrázkoch 6.1 a 6.2, tak najvýkonnejším prípravkom v tomto teste bol prípravok Nano Pi M4, ktorý dosahoval priepustnosti až jedného gigabajtu. V porovnaní s doskou, obsahujúcu rovnakým procesor avšak bez podpory šifrovania, Raspberry Pi sa jedná až o 20 násobné zväčšenie.

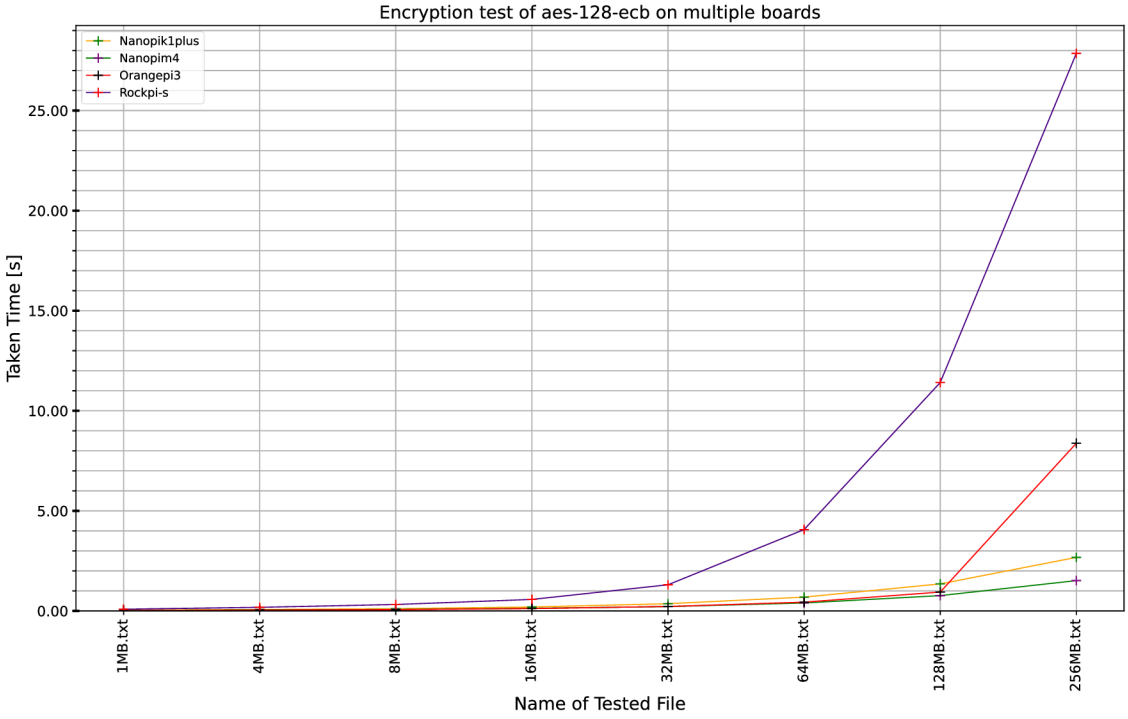
6.2 Šifrovanie pomocou encryptionscript-u

Na testovanie rýchlosti šifrovania, bol použitý skript *encryptionscript* (viď. Časť 5.4.1), ktorý šifroval textové súbory za použitia programu OpenSSL. Jedná sa o šifrovanie bez náhodného inicializačného vektoru s použitím rovnakého počiatočného hesla.

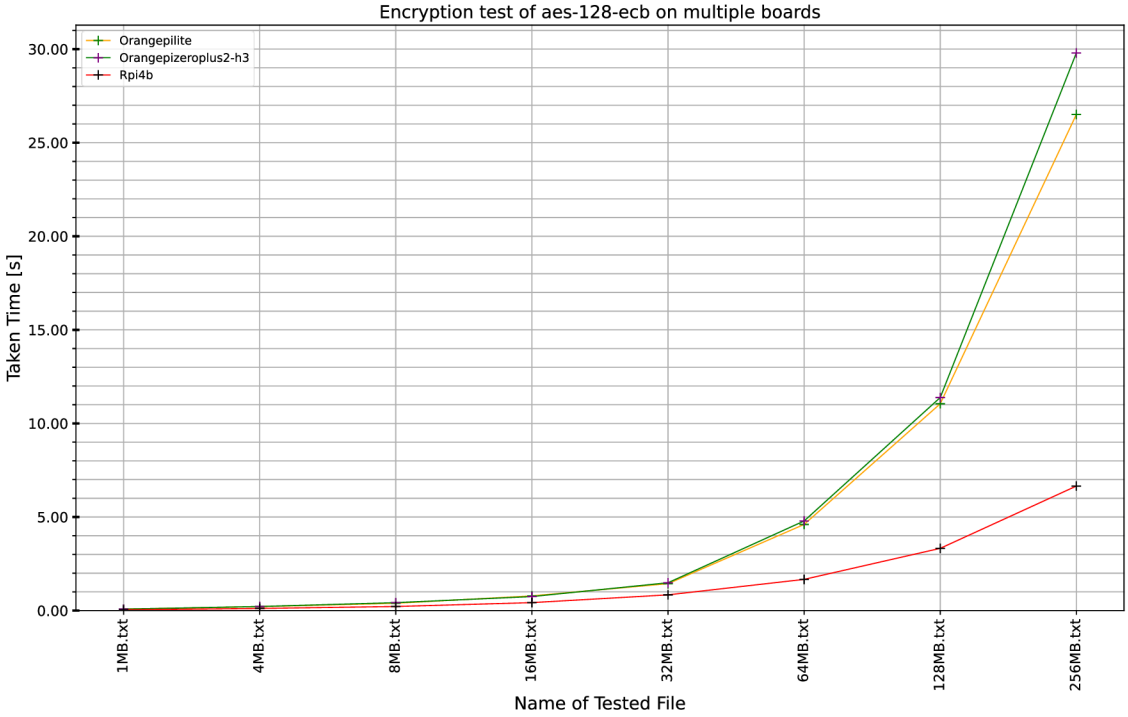
Tabuľka 6.2 Testovanie rýchlosti šifrovania súboru 16MB.txt pre operačné módy šifry AES-128

Šifra / Doska	AES-128-ECB	AES-128-CBC	AES-128-CFB	AES-128-OFB	AES-128-CTR
Raspberry Pi 4B	0,425379	0,431356	0,439982	0,435798	0,454595
Orange Pi Lite	0,785208	0,738975	0,773308	0,778323	0,758126
Orange Pi Zero +2 H3	0,748434	0,789040	0,767633	0,764054	0,801891
Nano Pi K1 Plus	0,196098	0,170540	0,205836	0,204660	0,172652
Nano Pi M4	0,129835	0,108098	0,121552	0,115886	0,121222
Orange Pi 3	0,117891	0,099164	0,126207	0,125341	0,100898
Rock Pi S	0,574516	0,839879	0,666827	0,748745	0,667968

V rámci prípravkou s podporou šifrovania (v tabuľke 6.2 zvýraznené tučne) je vidieť, že k najrýchlejšiemu šifrovaniu dochádza v prípravku Nano Pi M4 (v prípadoch operačných módov šifry CFB a OFB) a Orange Pi 3 (v prípadoch módov ECB, CBC a CTR). Najmenším rozdielom hodnôt v rámci tabuľky, je pri operačnom módu CFB práve medzi týmito doskami. Najväčší rozdiel hodnôt, môžeme pozorovať medzi doskami s podporov a bez podpory šifrovania Orange Pi 3 a Orange Pi lite (rozdiel v rýchlosti je viacej ako šesťnásobný).



Obrázok 6.3 Testovanie rýchlosti šifrovania, dosky s hardwarovou akceleráciou



Obrázok 6.4 Testovanie rýchlosti šifrovania, dosky bez hardwarovej akcelerácie

7. ZÁVER

Cieľom tejto práce bolo navrhnúť metriku testovania pre meranie kryptografického výkonu mikrokontrolerov z rady ARM Cortex-A a následne pomocou tohto návrhu vykonať testovanie.

V teoretickej časti práce, bolo vysvetlené fungovanie kryptografie, definovanie pojmov šifrovanie a dešifrovanie. Následné, bola kryptografia rozdelená na jednotlivé tipy šifrovacích a hashovacích systémov. Pri každom systéme, bolo vysvetlené jeho využitie a fungovanie. Kapitola 3 sa venovala predstaveniu architektúry ARM procesorov a ich inštrukčnej sady. Bola krátko opísaná, ako z historického tak i súčasného hľadiska. Taktiež, bolo zamerané na podporu v rámci kryptografie. Nasledujúca kapitola sa zaoberala nadefinovaním testovacích metrík a teoretickej prípravy testovacieho prostredia, ako zo softwarového tak hardwarového hľadiska.

V praktickej časti práce, boli opísané postupy pri procese prípravy testovacieho prostredia, či prípadne i procesy riešenia problémov ktoré sa vyskytli. V rámci kapitoly 5, boli vysvetlené dôvody, na zmenu používaného smerovania, popísaný proces a dôvod pre prechod na autentizáciu pomocou verejného kľúča v rámci protokolu SSH, proces inštalácie OS na hardwarový prípravok a následne ešte aj objasnenie fungovania naprogramovaných skriptov a programov, vytvorených pre testovanie hardwarových vzoriek. V kapitole 6 - Vyhodnocovanie zachytených údajov, sú analyzované a krátko porovnané zachytené údaje v rámci hardwarových prípravkou, ktorých procesor obsahuje kryptografickou akceleráciou šifrovania. Následne, sú tieto rezultáty porovnané s výsledkami testovania jednotiek ktoré tento modul neobsahujú.

Testovanie ukázalo, že rozšírenie obsahujúce hardwarovú akceleráciu kryptografie, výrazne urýchľuje šifrovanie pomocou AES z rádovo desiatok sekúnd až pol minúty na jednotky sekúnd. V hraničných prípadoch, je rozdiel medzi prípravkami s podporou a bez, až dvadsaťnásobný.

Ďalšie rozšírenie práce, sa môže uberať viacerými smermi. Jeden návrh, je pre pridanie testovania o meranie odberu elektrickej energie prípravkov, pomocou voltmetru počas šifrovania. Ako ďalší návrh, na rozšírenie, sa môže javiť vývoj grafického užívateľského rozhrania. Pomocou rozhrania, by sa užívateľ, mohol vzdialene pripájať k prípravkom pomocou SSH, získaval výsledky a vytváral grafy zachytených dát.

LITERATÚRA

[1] SHIREY, Robert *Internet Security Glossary, Version 2* [online]. Aug. 2007 [cit. 11.12.2021]. Dostupné z: https://datatracker.ietf.org/doc/html/rfc4949.txt

[2] KESSLER, Gary. *An Overview of Cryptography* [online]. [cit. 26.11.2021]. Dostupné z: https://www.garykessler.net/library/crypto.html

[3] STALLINGS, William a Lawrie BROWN. *Computer Security: Principles and Practice*. 2nd ed. Pearson Education, 2012. ISBN 978-0-13-277506-9.

[4] BURDA, Karel. *Aplikovaná kryptografie*. Brno: VUTIUM, 2013. ISBN 978-80-214-4612-0.

[5] BURDA, Karel. *Úvod do kryptografie*. Brno: Akademické nakladatelství CERM, 2015. ISBN 978-80-7204-925-7.

[6] CANTEAUT, Anne. Stream Cipher. *Encyclopedia of Cryptography and Security* [online]. Boston, MA: Springer US, 2011 [cit. 11.12.2021]. Dostupné z: https://doi.org/10.1007/978-1-4419-5906-5_374

[7] FONTAINE, Caroline. *Synchronous Stream Cipher*. Encyclopedia of Cryptography and Security [online]. Boston, MA: Springer US, 2011 [cit. 11.12.2021]. Dostupné z: https://doi.org/10.1007/978-1-4419-5906-5_376

[8] FONTAINE, Caroline. Self-Synchronizing Stream Cipher. *Encyclopedia of Cryptography and Security* [online]. Boston, MA: Springer US, 2011 [cit. 11.12.2021]. Dostupné z: https://doi.org/10.1007/978-1-4419-5906-5_371

[9] STALLINGS, William. *Cryptography and network security: principles and practice*. Seventh edition. Boston: Pearson, [2017]. Global edition. ISBN 978-1-292-15858-7.

[10] IBM CORPORATION. *Z/Transaction Processing Facility Enterprise Edition: Digital signatures* [online]. 2021 [cit. 12.12.2021]. Dostupné z: https://www.ibm.com/docs/en/ztpf/2021?topic=concepts-digital-signatures

[11] BURDA, Karel. *Kryptografie okolo nás*. Praha: CZ.NIC, z.s. p.o, 2019. ISBN 978-80-88168-49-2.

[12] MENEZES, Alfred J., Paul C.van OORSCHOT a Scott A. VANSTONE. *Handbook of applied cryptography*. Boca Raton: CRC Press, 1997. ISBN 0-8493-8523-7.

[13] PAAR, Christof, Jan PELZL a Bart PRENEEL. *Understanding Cryptography*. Berlin/Heidelberg: Springer Berlin / Heidelberg, 2010. ISBN 9783642041006.

[14] PRENEEL, Bart. Modes of Operation of a Block Cipher. *Encyclopedia of Cryptography and Security* [online]. Boston, MA: Springer US, 2011 [cit. 11.12.2021]. Dostupné z: https://doi.org/10.1007/978-1-4419-5906-5_599

[15] DWORKIN, Morris. NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY. *Recommendation for Block Cipher Modes of Operation* [online]. Washington, D.C.: U.S. Government Printing Office, 2001 [cit. 12.12.2021]. Dostupné z: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

[16] DWORKIN, Morris, Elaine BARKER, James NECHVATAL, Lawrence BASSHAM a James DRAY. *Federal Information Processing Standards Publication 197: Specification for the Advanced Encryption Standard (AES)* [online]. National Institute of Standards and Technology, 2001 [cit. 12.12.2021]. Dostupné z: https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf

[17] TŮMA, Jiří. *Konečná Tělesa* [online]. 2006 [cit. 12.12.2021]. Dostupné z: https://www2.karlin.mff.cuni.cz/~tuma/ffields/skripta.pdf

[18] WALSHE, Ben. *A Brief History of Arm: Part 1* [online]. 2015 [cit. 12.12.2021]. Dostupné z: https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/a-brief-history-of-arm-part-1

[19] WIKICHIP LLC. *ARMv1 - ARM - WikiChip* [online]. 2017 [cit. 12.12.2021]. Dostupné z: https://en.wikichip.org/wiki/arm/armv1

[20] ARM LIMITED. *ARM Architecture reference manual* [online]. E. 1996, 6.2000 [cit. 12.12.2021]. Dostupné z: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/third-party/ddi0100e_arm_arm.pdf

[21] WALSHE, Ben. *A Brief History of Arm: Part 2* [online]. 2015 [cit. 12.12.2021]. Dostupné z: https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/a-brief-history-of-arm-part-2

[22] CHEN, Crystal, Greg NOVICK a Kirk SHIMANO. *RICS Architecture: RISC vs. CISC* [online]. 2000 [cit. 12.12.2021]. Dostupné z: https://cs.stanford.edu/people/eroberts/courses/soco/projects/2000-01/risc/risccisc/index.html

[23] ARM LIMITED. *ARM Architecture Reference Manual ARMv7-A and ARMv7-R edition* [online]. C.d. Cambridge, England: Arm Limited, 2007, 29.3.2018 [cit. 12.12.2021]. Dostupné z: https://developer.arm.com/documentation/ddi0406/latest

[24] ARM LIMITED. *ARM Cortex-A Series Programmer's Guide for ARMv7-A* [online]. 4. Cambridge, England: Arm Limited, 2013, 22.1.2014 [cit. 12.12.2021]. Dostupné z: https://developer.arm.com/documentation/den0013/d/ARM-Architecture-and-Processors/Architectural-profiles

[25] ARM LIMITED. *Arm Architecture Reference Manual Armv8, for Armv8-A architecture profile* [online]. G.b. Cambridge, England: Arm Limited, 2013, 22.7.2021 [cit. 12.12.2021]. Dostupné z: https://developer.arm.com/documentation/ddi0487/gb/?lang=en

[26] SILICON LABORATORIES. *Which ARM Cortex Core Is Right for Your Application* [online]. 2020 [cit. 12.12.2021]. Dostupné z: https://www.silabs.com/documents/public/white-papers/Which-ARM-Cortex-Core-Is-Right-for-Your-Application.pdf

[27] RIKEN. *Japan's Fugaku gains title as world's fastest supercomputer* [online]. 2020 [cit. 12.12.2021]. Dostupné z: https://www.riken.jp/en/news_pubs/news/2020/20200623_1/
[28] ARM LIMITED. *IoT Solutions– Internet of Things Solutions* [online]. [cit. 2022-05-26]. Dostupné z: https://www.arm.com/solutions/iot
[29] NEVONSOLUTIONS PVT. LTD. *Nevon Projects: ARM Cortex & ARM 7 Projects* [online]. [cit. 2022-05-26]. Dostupné z: https://nevonprojects.com/arm-microcontroller-projects/
[30] ARM LIMITED. *ARM Cortex-A57 MPCore Processor Technical Reference Manual* [online]. H. Cambridge, England: Arm Limited, 2013, 1.2.2016 [cit. 12.12.2021]. Dostupné z: https://developer.arm.com/documentation/ddi0488/h
[31] RASPBERRY PI. *Raspberry Pi Documentation: BCM2711* [online]. 2021 [cit. 12.12.2021]. Dostupné z: https://www.raspberrypi.com/documentation/computers/processors.html#bcm2711
[32] SHENZHEN XUNLONG SOFTWARE. *Orange Pi: Orange Pi Lite* [online]. ©2016 [cit. 12.12.2021]. Dostupné z: http://www.orangepi.org/orangepilite/
[33] SHENZHEN XUNLONG SOFTWARE. *Orange Pi: Orange Pi Zero Plus2* [online]. ©2016 [cit. 12.12.2021]. Dostupné z: http://www.orangepi.org/OrangePiZeroPlus2/
[34] FRIENDLYARM. *FriendlyARM WiKi: NanoPi K1 Plus* [online]. 2021, 1.12.2021 [cit. 12.12.2021]. Dostupné z: https://wiki.friendlyarm.com/wiki/index.php/NanoPi_K1_Plus
[35] FRIENDLYARM. *FriendlyARM WiKi: NanoPi M4* [online]. 2021, 02.29.2021 [cit. 12.12.2021]. Dostupné z: https://wiki.friendlyarm.com/wiki/index.php/NanoPi_M4
[36] SHENZHEN XUNLONG SOFTWARE. *Orange Pi 3 User Manual* [online]. 2019 [cit. 12.12.2021]. Dostupné z: https://drive.google.com/file/d/1iURY6cLZhFewrCrn37UCVoORnENuAoDK/view?usp=sharing
[37] RADXA LIMITED. *Radxa Wiki: Rock Pi S* [online]. 2021 [cit. 12.12.2021]. Dostupné z: https://wiki.radxa.com/RockpiS/hardware/rockpiS
[38] ALLWINNER TECHNOLOGY. *Allwinner H3 Datasheet* [online]. April 25,2015 [cit. 11.12.2021]. Dostupné z: https://linux-sunxi.org/images/4/4b/Allwinner_H3_Datasheet_V1.2.pdf
[39] ALLWINNER TECHNOLOGY. *Allwinner H5 Datasheet* [online]. May 20,2016 [cit. 11.12.2021]. Dostupné z: https://linux-sunxi.org/images/a/a3/Allwinner_H5_Manual_v1.0.pdf
[40] FUZHOU ROCKCHIP ELECTRONICS. *Rockchip RK3308 Datasheet* [online]. Feb. 27,2018 [cit. 11.12.2021]. Dostupné z: https://dl.radxa.com/rockpis/docs/hw/datasheets/Rockchip%20RK3308%20Datasheet%20V1.0-2018027.pdf

[41] ARM LIMITED. *Technologies: Arm Big.little* [online]. [cit. 11.12.2021]. Dostupné z: https://www.arm.com/why-arm/technologies/big-little
[42] FUZHOU ROCKCHIP ELECTRONICS. *Rockchip RK3399 Datasheet* [online]. May 29,2018 [cit. 11.12.2021]. Dostupné z: https://www.rockchip.fr/RK3399%20datasheet%20V1.8.pdf
[43] ALLWINNER TECHNOLOGY. *Allwinner H6 V200 User Manual* [online]. Oct. 17,2017 [cit. 11.12.2021]. Dostupné z: https://linux-sunxi.org/images/4/46/Allwinner_H6_V200_User_Manual_V1.1.pdf
[44] OPENSSL SOFTWARE FOUNDATION. *OpenSSL Project* [online]. [cit. 11.12.2021]. Dostupné z: https://www.openssl.org/
[45] THE OPENBSD PROJECT. *OpenSSH: Project History* [online]. [cit. 11.12.2021]. Dostupné z: https://www.openssh.com/history.html
[46] TATHAM, Simon. *PuTTY User Manual* [online]. [cit. 11.12.2021]. Dostupné z: https://the.earth.li/~sgtatham/putty/0.76/htmldoc/
[47] BALENA. *BalenaEtcher: Flash OS images to SD cards & USB drives* [online]. 2017 [cit. 2022-05-20]. Dostupné z: https://www.balena.io/etcher/?

ZOZNAM SYMBOLOV A SKRATIEK

Skratky:

AES	Advanced Encryption Standard (Štandard pokročilého šifrovania)
ARM	Advanced RISC Machine (Pokročilý RISC procesor)
CBC	Cipher block chaining (Reťazenie šifrových blokov)
CFB	Cipher feedback (Šifrová spätná väzba)
CISC	Complex instruction-set computer (Počítač s rozsiahlou inštrukčnou sadou)
CPU	Central processing unit (Centrálna procesorová jednotka)
CTR	Counter Mode (Výstupná spätná väzba)
DES	Digital Encryption Standart (Štandard digitálneho šifrovania)
ECB	Electornic code book (Elektronická kódová kniha)
OFB	Output feedback (Výstupná spätná väzba)
RISC	Reduced instruction-set computer (Počítač s obmedzenou sadou inštrukcií)
SBC	Single Board Computer (Jednodoskový počítač)
S-box	Substitution box (Substitučná tabuľka)
SHA	Secure Hash Algorithms (Bezpečné hashovacie algoritmy)
SoC	System on a chip (Systém na čipe)
TDES	Triple Data Encryption Algorithm (Algoritmus trojitého šifrovania údajov)
XOR	Exclusive disjunction (Vylučujúca disjunkcia)
SSH	Secure Shell (Zabezpečený prístup k príkazovému interpretovaču)
DHCP	Dynamic Host Configuration Protocol (Protokol dynamickej konfigurácie hostiteľa)
SCP	Secure copy protocol (Protokol zabezpečeného kopírovania)
IP	Internet Protocol (Internetový protokol)
CSV	Comma-separated values (Hodnoty oddelené čiarkami)
SVG	Scalable Vector Graphics (Škálovateľná vektorová grafika)

Symboly:

C	zašifrovaný text	(-)
E_k	šifrovanie s použitím kľúču	(-)
K	kľuč použitý v šifrovaní	(-)
M	nezašifrovaná správa	(-)
D_k	dešifrovanie s použitím kľúču	(-)

ZOZNAM PRÍLOH

Príloha A – Zdrojové kódy skriptov a programov ktoré boli použité v rámci testovania, testovacie dáta, zachytené údaje a spracované grafy sa nachádzajú na priloženom CD.