

Návrh nové symetrické šifry pro mobilní zařízení

Bc. Petr Žáček

Diplomová práce
2014



Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
akademický rok: 2013/2014

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Petr Žáček**
Osobní číslo: **A12336**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Bezpečnostní technologie, systémy a management**
Forma studia: **prezenční**

Téma práce: **Návrh nové symetrické šifry pro mobilní zařízení**
Téma anglicky: **Designing New Symetric Ciphers for Mobile Devices**

Zásady pro vypracování:

1. Navrhněte blokovou symetrickou šifru.
2. Analyzujte bezpečnost a rychlost navrženého symetrického algoritmu pro šifrování a jeho porovnání se známými symetrickými šifrovacími algoritmy DES, 3DES a AES.
3. Implementujte navrženou šifru v programovacím jazyce Python 3.x.
4. Vytvořte grafické uživatelské rozhraní pro použití navržené šifry k šifrování/dešifrování souborů v jazyce Python 3.x.
5. Zhodnoťte využitelnost navržené symetrické šifry pro mobilní zařízení.

Rozsah diplomové práce:

Rozsah příloh:

Forma zpracování diplomové práce: tištěná/elektronická

Seznam odborné literatury:

1. VONDRUŠKA, Pavel. Kryptologie, šifrování a tajná písma. 1. vyd. Praha: Albatros, 2006, 340 s. ISBN 80-00-01888-8.
2. SINGH, Simon. Kniha kódů a šifer: tajná komunikace od starého Egypta po kvantovou kryptografii. 2. vyd. v českém jazyce. Praha: Dokořán, 2009, 382 s. ISBN 978-80-7363-268-7.
3. Python 3.3.3 documentation [online]. 2014 [cit. 2014-02-05]. Dostupné z: <http://docs.python.org/3/>.
4. SUMMERFIELD, Mark. Python 3: Výukový kurz. Vyd. 1. Překlad Lukáš Krejčí. Brno: Computer Press, 2010, 584 s. ISBN 978-80-251-2737-7.
5. ŠENKERÍK, Roman. Přednášky z předmětu Kryptologie. 2005-2013.

Vedoucí diplomové práce:

Ing. David Malaník, Ph.D.

Ústav informatiky a umělé inteligence

Datum zadání diplomové práce:

7. února 2014

Termín odevzdání diplomové práce:

27. května 2014

Ve Zlíně dne 7. února 2014

prof. Ing. Vladimír Vašek, CSc.
děkan



doc. RNDr. Vojtěch Křesálek, CSc.
ředitel ústavu

Prohlašuji, že

- beru na vědomí, že odevzdáním diplomové/bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že diplomová/bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji diplomovou/bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen s předchozím písemným souhlasem Univerzity Tomáše Bati ve Zlíně, která je oprávněna v takovém případě ode mne požadovat přiměřený příspěvek na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše);
- beru na vědomí, že pokud bylo k vypracování diplomové/bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky diplomové/bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem diplomové/bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na diplomové práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze diplomové práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně

.....
podpis diplomanta

ABSTRAKT

Tato diplomová práce se zabývá návrhem nové symetrické blokové šifry a její implementaci v programovacím jazyce Python verze 3.x. V teoretické části jsou rozebrány možnosti symetrické blokové kryptografie a dnešní nejznámější blokové šifry. V praktické části je popsán návrh nové symetrické blokové šifry – její součástí a vlastností. Dále je provedena analýza rychlosti a výkonnost navržené šifry spolu s vyvozením závěrů. V další části se navržená šifra analyzuje z pohledu bezpečnosti a je proveden rozbor výsledku z analýzy. Část práce se věnuje popisu implementace grafického uživatelské rozhraní pro šifrování/dešifrování souborů a na závěr se diskutuje o možnostech použití navržené šifry na mobilních zařízeních.

Klíčová slova:

Bloková šifra, symetrická kryptografie, Python, proměnlivá struktura

ABSTRACT

This master thesis deals with design of the new symmetric block cipher and its implementation in programming language Python version 3.x. In the theoretical part are discussed options of symmetric block cryptography and description of the most common ciphers of these days. In the practical part is described design of the new symmetric block cipher – its parts and properties. Speed and efficiency of designed cipher were tested in another part including of conclusions of it. Next the designed cipher was analyzed for its Security and then the results of analysis are written and analyzed, too. The next part is about implementation of graphical user interface for encryption/decryption of files and the possibilities of using of designed cipher are discussed in the end of this work.

Keywords:

Block cipher, symmetric cryptography, Python, variable structure

Touto cestou bych chtěl moc poděkovat zejména vedoucímu mé diplomové práce Ing. Davidu Malaníkovi, Ph.D. za cenné rady a konzultace ohledně diplomové práce. Dále bych chtěl poděkovat své přítelkyni a své rodině za toleranci a psychickou podporu v čase vypracovávání diplomové práce.

OBSAH

ÚVOD.....	11
TEORETICKÁ ČÁST.....	13
1.1 MODERNÍ SYMETRICKÁ KRYPTOGRAFIE	15
1.2 BLOKOVÉ ŠIFRY	16
1.2.1 OPERACE BLOKOVÝCH ŠIFER.....	16
1.2.1.1 Operace XOR.....	16
1.2.1.2 Operace sčítání modulo.....	17
1.2.1.3 Operace násobení modulo.....	21
1.2.1.4 Substituce a transpozice	21
1.2.2 REŽIMY ČINNOSTI – SPRÁVA KLÍČE (KEY-MANAGEMENT).....	22
1.2.2.1 Režim činnosti ECB – Electronic Code Book	22
1.2.2.2 Režim činnosti CBC – Cipher Block Chaining.....	23
1.2.2.3 Režim činnosti OFB – Output Feedback Block.....	23
1.2.2.4 Režim činnosti CFB – Cipher Feedback Block	24
1.2.3 BLOKOVÁ ŠIFRA DES.....	25
1.2.4 BLOKOVÁ ŠIFRA 3DES (TRIPLE-DES)	26
1.2.5 BLOKOVÁ ŠIFRA AES.....	26
2.1 BEZPEČNOST NEBO RYCHLOST	30
2.2 DÉLKA KLÍČE.....	31
2.3 OPERACE PRO ŠIFROVÁNÍ/DEŠIFROVÁNÍ	32
2.3.1 VELIKOST BLOKU	33
2.3.2 REŽIM ČINNOSTI – SPRÁVA KLÍČE (KEY-MANAGEMENT)	33
2.4 STRUKTURA ŠIFROVÁNÍ/DEŠIFROVÁNÍ A POČET RUND	33
2.5 TVORBA INICIALIZAČNÍHO VEKTORU	33
2.6 DOPLNĚNÍ NECELÉHO BLOKU.....	34
PRAKTICKÁ ČÁST	35
3.1 ZÁKLADNÍ PRVKY A VLASTNOSTI NAVRŽENÉ BLOKOVÉ ŠIFRY	36
3.2 GENEROVÁNÍ INICIALIZAČNÍHO VEKTORU Z UŽIVATELSKÉHO HESLA A HASHE INICIALIZAČNÍHO VEKTORU.....	42
3.2.1 KROK 1 – PRODLOUŽENÍ UŽIVATELSKÉHO HESLA	43
3.2.2 KROK 2 – PŘIPOJENÍ DEFAULTNÍHO ŘETĚZCE	43
3.2.3 KROK 3 – VMÍSENÍ ZNAKŮ DEFAULTNÍHO ŘETĚZCE	44
3.2.4 KROK 4 – VÝPOČET HASHE HESLA PO VMÍSENÍ ZNAKŮ DEFAULTNÍHO ŘETĚZCE	44
3.2.5 KROK 5 – GENEROVÁNÍ <i>IV</i> NA ZÁKLADĚ KOMBINOVÁNÍ HASHE HESLA A HESLA PO VMÍSENÍ ZNAKŮ DEFAULTNÍHO ŘETĚZCE	44
3.2.5.1 Zdrojový kód pro Krok 5 v jazyce Python verze 3.x	47
3.2.6 KROK 6 – VÝPOČET HASHE	48
3.3 NÁVRH POMOCNÉ FUNKCE PRO ZAMÍCHÁNÍ SUBSTITUČNÍ TABULKY	48

3.3.1	ZDROJOVÝ KÓD FUNKCE PRO ZAMÍCHÁNÍ SUBSTITUČNÍ TABULKY	49
3.4	NÁVRH OPERACÍ PRO ŠIFROVÁNÍ	49
3.4.1	SUBSTITUTE	50
3.4.1.1	Zdrojový kód funkce pro substituci	50
3.4.2	OPERACE ADD – PŘÍČTENÍ KLÍČE K DATŮM	51
3.4.2.1	Zdrojový kód funkce pro Add – přičtení klíče k datům	51
3.4.3	MIXOVÁNÍ - TRANSPOZICE	51
3.4.3.1	Zdrojový kód funkce pro mixování	52
3.5	NÁVRH OPERACÍ PRO DEŠIFROVÁNÍ	52
3.5.1	„DeSUBSTITUTE“ – INVERZNÍ OPERACE K SUBSTITUCI	52
3.5.1.1	Zdrojový kód funkce pro inverzní operaci k substituci	53
3.5.2	„DeADD“ – INVERZNÍ OPERACE K ADD – ODEČTENÍ KLÍČE OD DAT	53
3.5.2.1	Zdrojový kód funkce pro inverzní operaci k Add – odečtení klíče	53
3.5.3	„DeMIXOVÁNÍ“ – INVERZNÍ OPERACE K MIXOVÁNÍ	54
3.5.3.1	Zdrojový kód funkce pro inverzní operaci k mixování	54
3.6	OPERACE PRO URČENÍ POSLOUPNOSTI OPERACÍ PRO ŠIFROVÁNÍ A DEŠIFROVÁNÍ	54
3.6.1	ZDROJOVÝ KÓD FUNKCE PRO VYGENEROVÁNÍ POSLOUPNOSTI OPERACÍ	56
3.7	REŽIM ČINNOSTI – OPERACE PRO POSUN KLÍČE	57
3.7.1	ZDROJOVÝ KÓD FUNKCE PRO POSUNUTÍ KLÍČE	58
3.8	POMOCNÁ FUNKCE PRO ZAKÓDOVÁNÍ BAJTU UDÁVAJÍCÍ POČET DOPLNĚNÝCH ZNAKŮ (BAJTŮ) NECELÉHO BLOKU DAT	59
3.8.1	ZDROJOVÝ KÓD FUNKCE PRO ZAKÓDOVÁNÍ BAJTU UDÁVAJÍCÍ POČET DOPLNĚNÝCH ZNAKŮ	59
3.9	FUNKCE PRO DEKÓDOVÁNÍ BAJTU UDÁVAJÍCÍHO POČET DOPLNĚNÝCH ZNAKŮ (BAJTŮ) NECELÉHO BLOKU DAT	60
3.9.1	ZDROJOVÝ KÓD FUNKCE PRO DEKÓDOVÁNÍ BAJTU UDÁVAJÍCÍ POČET DOPLNĚNÝCH ZNAKŮ	60
3.10	FUNKCE PRO ZAKÓDOVÁNÍ POČTU RUND	61
3.10.1	ZDROJOVÝ KÓD FUNKCE PRO ZAKÓDOVÁNÍ POČTU RUND	61
3.11	FUNKCE PRO DEKÓDOVÁNÍ POČTU RUND	62
3.11.1	ZDROJOVÝ KÓD FUNKCE PRO DEKÓDOVÁNÍ POČTU RUND	62
3.12	VELIKOST ZAŠIFROVANÉHO SOUBORU S PŘÍPONOU .ENCRYPT	62
4.1	VÝPOČETNÍ SLOŽITOSTI A DOBY TRVÁNÍ OPERACÍ PRO ŠIFROVÁNÍ	64
4.1.1	VÝPOČETNÍ SLOŽITOST A DOBA TRVÁNÍ OPERACE SUBSTITUTE	65
4.1.2	VÝPOČETNÍ SLOŽITOST A DOBA TRVÁNÍ OPERACE PŘÍČTENÍ KLÍČE – ADD	67
4.1.3	VÝPOČETNÍ SLOŽITOST A DOBA TRVÁNÍ OPERACE MIXOVÁNÍ	70
4.2	VÝPOČETNÍ SLOŽITOSTI A DOBY TRVÁNÍ OPERACÍ PRO DEŠIFROVÁNÍ	72

4.2.1	VÝPOČETNÍ SLOŽITOST A DOBA TRVÁNÍ „DeSUBSTITUTE“ – INVERZNÍ OPERACE K SUBSTITUCI	72
4.2.2	VÝPOČETNÍ SLOŽITOST A DOBA TRVÁNÍ „DeADD“ – OPERACE ODEČTENÍ KLÍČE OD DAT	74
4.2.3	VÝPOČETNÍ SLOŽITOST A DOBA TRVÁNÍ „DeMIXOVÁNÍ“ – INVERZNÍ OPERACE K MIXOVÁNÍ	76
4.3	SROVNÁNÍ OPERACÍ PRO ŠIFROVÁNÍ A DEŠIFROVÁNÍ	78
4.4	VÝPOČETNÍ SLOŽITOST A DOBY TRVÁNÍ OPERACÍ PRO REŽII.....	79
4.4.1	VÝPOČETNÍ SLOŽITOST A DOBA TRVÁNÍ OPERACE PRO ZAMÍCHÁNÍ SUBSTITUČNÍ TABULKY	79
4.4.2	VÝPOČETNÍ SLOŽITOST A DOBA TRVÁNÍ OPERACE PRO POSUNUTÍ KLÍČE	81
4.4.3	VÝPOČETNÍ SLOŽITOST A DOBA TRVÁNÍ OPERACE PRO GENEROVÁNÍ POSLOUPNOSTI OPERACÍ	83
4.4.4	VÝPOČETNÍ SLOŽITOST A DOBA TRVÁNÍ OPERACE GENEROVÁNÍ <i>IV</i>	85
4.5	VÝKLYKY DOBY TRVÁNÍ OPERACÍ PRO ŠIFROVÁNÍ/DEŠIFROVÁNÍ A REŽIJNÍCH OPERACÍ.....	88
4.6	DOBA TRVÁNÍ ŠIFROVÁNÍ/DEŠIFROVÁNÍ SOUBORU V ZÁVISLOSTI NA JEHO VELIKOSTI	88
4.7	VLIV HARDWAROVÉ KONFIGURACE NA DOBU ŠIFROVÁNÍ/DEŠIFROVÁNÍ SOUBORU.....	90
4.8	POROVNÁNÍ DOBY POTŘEBNÉ K ŠIFROVÁNÍ NAVRŽENÉ ŠIFRY S ŠIFRAMI AES, DES A 3DES	97
4.9	MOŽNOSTI VYLEPŠENÍ RYCHLOSTI NAVRŽENÉ ŠIFRY.....	98
5.1	SHRnutí VLASTNOSTÍ ŠIFRY A CO TO ZNAMENÁ PRO BEZPEČNOST	100
5.1.1	DĚLKA KLÍČE	100
5.1.2	GENEROVÁNÍ <i>IV</i> Z UŽIVATELSKÉHO HESLA PRO ŠIFROVÁNÍ	100
5.1.3	POUŽITÉ OPERACE PRO ŠIFROVÁNÍ	102
5.1.4	GENEROVÁNÍ POSLOUPNOSTI OPERACÍ PRO ŠIFROVÁNÍ A POČET RUND.....	103
5.1.4.1	Otestování vlivu jiné posloupnosti operací na vstupní data	104
5.1.5	POSUNUTÍ KLÍČE – REŽIM ČINNOSTI.....	105
5.1.6	ZAKÓDOVÁNÍ BAJTU UDÁVAJÍCÍ POČET DOPLNĚNÝCH ZNAKŮ (BAJTŮ) NECELÉHO BLOKU	106
5.1.7	ZAKÓDOVÁNÍ POČTU RUND	107
5.1.8	ABSENCE RUNDOVNÍCH KLÍČŮ	107
5.1.9	ZAMÍCHÁNÍ SUBSTITUČNÍ TABULKY	108
5.1.10	OZNAČENÍ VÝSTUPNÍHO SOUBORU PŘÍPONOU .ENCRYPT	108
5.1.11	SHRnutí A ZÁVĚRY	109
5.2	VÝPOČET LAVINOVÉHO EFEKTU PRO NAVRŽENOU ŠIFRU	109

5.2.1	LAVINOVÝ EFEKT JEDNOHO VSTUPNÍHO BLOKU PŘI ZMĚNĚ JEDNOHO BITU - ŠIFROVÁNÍ JEDNOU RUNDOU	110
5.2.2	LAVINOVÝ EFEKT JEDNOHO VSTUPNÍHO BLOKU BEZE ZMĚNY BITŮ - ŠIFROVÁNÍ VE VÍCE RUNDÁCH	114
5.2.3	LAVINOVÝ EFEKT DVOU PO SOBĚ JDOUCÍCH KLÍČŮ	115
5.2.4	LAVINOVÝ EFEKT DVOU PO SOBĚ ZAŠIFROVANÝCH BLOKŮ STEJNÝCH PŘED ŠIFROVÁNÍM	115
5.2.5	LAVINOVÝ EFEKT PŘI GENEROVÁNÍ <i>IV</i> ZE ZADANÉHO HESLA	116
5.2.6	LAVINOVÝ EFEKT PŘI ZMĚNĚ JEDNOHO BITU VSTUPNÍHO HESLA	116
5.2.7	LAVINOVÝ EFEKT V ZÁVISLOSTI NA POUŽITÉ POSLOUPNOSTI OPERACÍ.....	117
5.2.8	ZÁVĚRY Z ANALÝZY LAVINOVÉHO EFEKTU	117
5.2.9	ZDROJOVÝ KÓD FUNKCE PRO VÝPOČET LAVINOVÉHO EFEKTU.....	118
5.3	KRYPTOANALYTICKÉ METODY	118
5.3.1	FREKVENČNÍ ANALÝZA	118
5.3.1.1	Zdrojový kód funkce pro výpočet četnosti znaků výše uvedené frekvenční analýzy.	120
5.3.2	INDEX KOINCIDENCE	120
5.3.3	LINEÁRNÍ A DIFERENCIÁLNÍ KRYPTOANALÝZA.....	120
5.3.4	JINÉ POKROČILÉ KRYPTOANALYTICKÉ METODY	121
5.3.5	ÚTOK HRUBOU SILOU	121
5.4	POSOUZENÍ BEZPEČNOSTI S ŠIFRAMI DES, 3DES A AES	121
5.4.1	NAVRŽENÁ ŠIFRA VERSUS DES	121
5.4.2	NAVRŽENÁ ŠIFRA VERSUS 3DES	122
5.4.3	NAVRŽENÁ ŠIFRA VERSUS AES	122
5.5	ZHODNOCENÍ ZÁVĚRŮ Z ANALÝZY BEZPEČNOSTI	123
6.1	GUI – HLAVNÍ OKNO	125
6.2	PODOKNO PRO PRŮBĚH ŠIFROVÁNÍ/DEŠIFROVÁNÍ.....	127
6.3	PODOKNO PRO CHYBOVÉ HLÁŠENÍ.....	128
6.4	PODOKNO PRO FUNKCI BENCHMARK	129
6.5	MOŽNOSTI ZLEPŠENÍ A ROZŠÍŘENÍ FUNKCÍ GUI.....	129
7.1	MOŽNOSTI POUŽITÍ NAVRŽENÉ ŠIFRY NA MOBILNÍCH ZAŘÍZENÍ	131
	SEZNAM POUŽITÉ LITERATURY.....	135
	SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK.....	136
	SEZNAM OBRÁZKŮ	137
	SEZNAM TABULEK.....	141
	SEZNAM PŘÍLOH.....	142

ÚVOD

Žijeme ve 21. století, v digitálním světě. Technika je všude kolem nás a denně na ni narážíme. Troufám si tvrdit, že minimálně 80 procent lidí v České republice vlastní nebo pracuje s výpočetní technikou, která nám každodenně ulehčuje práci, a bereme ji jako všední záležitost. S neustálým rozvojem mobilních technologií musíme zabezpečovat své soukromí i v tomto případě.

Mezi nejpoužívanější šifry moderní kryptografie jsou navrženy před více jak 20 lety. Jedná se o šifry AES, 3DES. Šifry nebyly doposud prolomené, ale je to jen otázkou času, kdy k tomu dojde. V ten moment bude potřeba nová a inovativní šifra, která by odolala nejmodernějším kryptoanalytickým metodám. Minimálně proto stojí za pokus zkusit navrhnout novou šifru.

Osobně jsem si zvolil toto téma nejen proto, že mě tato tematika baví a věnuji se jí i ve volném čase, ale i proto, že je otevřená novým možnostem. Leckdo může namítnout, že navrhnout novou šifru není potřeba, že AES je neprolomitelný, ale kdyby k tomu došlo, znamenalo by to katastrofu. Na šifře AES a dalších jsou závislé skoro všechny systémy světa. V oblasti mobilních technologií se do nedávna používaly jen proudové šifry, ale s příchodem smartphonu, které umožňují více než jen volat a posílat zprávy, přišla nutnost šifrovat i další obsah mobilních zařízení. Máme zde tablety, které brzy spolu s notebooky možná nahradí stolní počítače, protože jsou čím dál víc výkonnější. A proto i použití blokových šifer je čím dál běžnější i pro mobilní zařízení.

Dlouho zavedené stereotypy mohou vést ke katastrofě. I to je jeden z důvodů, s kterým se pokouší tato diplomová práce vypořádat. Práce se zaměřuje na návrh nové blokové šifry, která by mohla být použitelná i pro mobilní zařízení. V teoretické části budou rozebrány nejpoužívanější šifry a metody k šifrování dnešní doby – úvod do moderní kryptografie. Dále bude popsán návrh šifry teoreticky, které prvky zvolit a proč. V praktické části bude popsán návrh nové blokové šifry a její všechny prvky a vlastnosti. Dále bude rozebrána výpočetní složitost a rychlost navržené šifry a rezervy, které v tomto ohledu šifra má. Následně bude zhodnocena bezpečnost šifry, včetně všech prvků šifry. Šifra bude otestovaná na lavinový efekt, který je důležitým prvkem bezpečnosti. Nakonec bude analýza bezpečnosti zhodnocena a případně budou navrženy opatření k eliminaci zjištěných hrozeb. Šifra bude naimplementována v programovacím jazyce Python verze 3.x, a proto bude ukázán zdrojový kód všech navržených prvků a části šifry. A to včetně

grafického uživatelského rozhraní, které umožní vyzkoušet šifrování a dešifrování souborů pomocí navržené šifry. Na závěr bude zhodnoceno využití navržené šifry pro mobilní zařízení.

I. TEORETICKÁ ČÁST

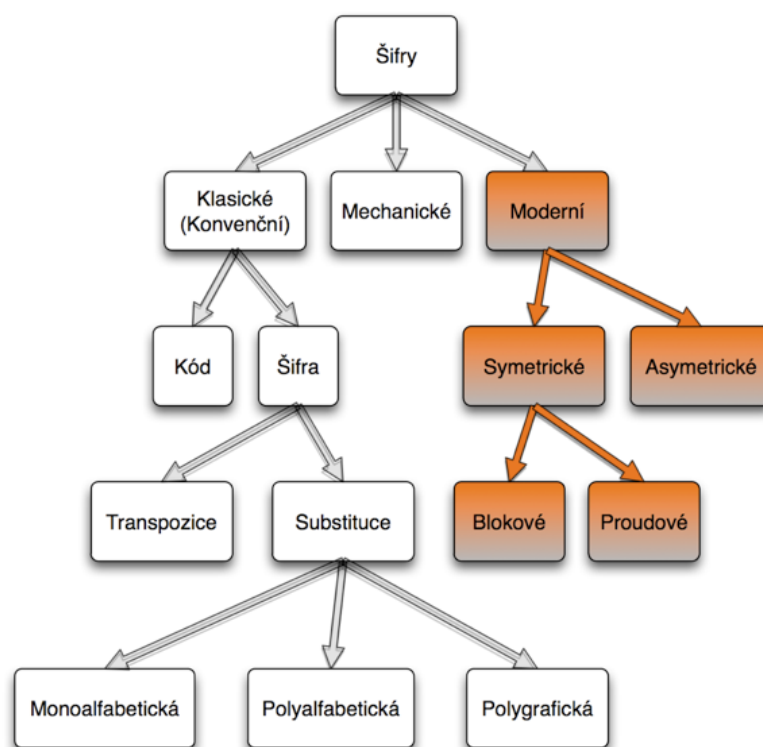
1 ÚVOD DO MODERNÍ KRYPTOGRAFIE

Na začátek je vysvětleno několik důležitých pojmů, které v následující práci budou používány.

Kryptografie se zabývá matematickými metodami vztahujícími se k takovým prvkům informační bezpečnosti, jako je zajištění důvěryhodnosti zprávy, integrity dat, autentizace entit a původu dat, včetně zkoumání jejich silných stránek a slabin i odolnosti vůči různým metodám útoků. Kryptografie je spolu s kryptoanalýzou a steganografií součástí vědy, kterou nazýváme kryptologie.[1]

Moderní ji nazýváme z důvodu, protože je využívána v informačních technologiích (počítače, mobilní telefony a jiná výpočetní zařízení). Na rozdíl od tzv. konvenční kryptografie, která je taky symetrická, nelze šifrování provést za použití tužky a papíru. Moderní kryptografii datujeme přibližně od roku 1917.[5]

Rozdělení kryptografie je patrné z následujícího obrázku, kde oranžová větev ukazuje moderní část kryptografie.

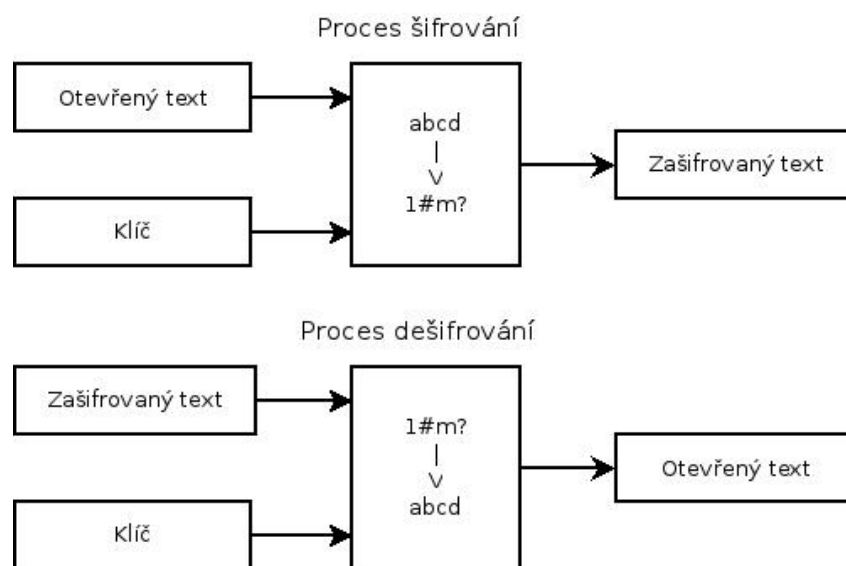


Obrázek 1 - Členění kryptografie [5]

Moderní kryptografii dělíme na symetrickou, asymetrickou a hybridní. Asymetrická kryptografie potřebuje k šifrování/dešifrování tzv. klíčový pár – veřejný a soukromý klíč. Veřejným klíčem zprávu šifrujeme a soukromým dešifrujeme. Mezi hlavní zástupce patří šifra RSA. Dále se věnovat asymetrické kryptografii nebudeme, i když je stejně fascinující jako symetrická kryptografie. Hybridní kryptografie kombinuje prvky a výhody kryptografie asymetrické a symetrické. Obvykle se data šifrují symetrickou kryptografií a klíč pro symetrickou kryptografii se zašifruje pomocí asymetrické kryptografie. [5]

1.1 Moderní symetrická kryptografie

Název symetrická kryptografie je odvozena od principu její činnosti. Pro šifrování i pro dešifrování používá stejný klíč. Princip činnosti znázorňuje následující obrázek.



Obrázek 2 - Blokové schéma principu symetrické kryptografie

Moderní symetrickou kryptografii dělíme do dvou skupin – proudové a blokové šifry. Proudové šifry šifrují bit po bitu, v reálném čase. Pokud přicházejí data, rozšiřuje se klíč. Využívají se všude tam, kde se předem neví, jak velká data budou potřeba šifrovat/dešifrovat. Proto se dnes využívají například k šifrování hovorů a datových přenosů v mobilní komunikaci, wi-fi komunikaci a datastreamech. Obvykle se šifruje/dešifruje pomocí funkce XOR (exkluzivní OR, exkluzivní disjunkce). [5] Dále se proudovým šifrům věnovat nebudeme, protože nejsou náplní práce, budeme se zabývat pouze návrhem a tvorbě blokové šifry.

1.2 Blokové šifry

Blokovým šifrářem se v této části budeme věnovat podrobněji. Jak vyplývá z názvu, jedná se o šifry, které šifrují/dešifrují celé bloky dat o pevně dané délce. Bloky, které jsou kratší (koncové bloky), se doplní o počet chybějících znaků. [5] Jednotlivé šifry se od sebe liší zejména v následujících bodech.

- Velikost bloku – 64, 128, 192, 256, 312, 512 (obvykle násobky čísla 8 → bajt, 8 bitů)
- Délka klíče – často stejná délka, jako je délka bloku
- Operace prováděné nad bloky – XOR, sečtení modulo, násobení modulo, substituce, transpozice (permutace), bitový posun
- Počet rund (provedení sledu operací nad blokem) – 8.5, 10, 12, 14, 16
- Struktura rundy, poskládání operací
- Principem generování inicializačního vektoru
- Režimem činnosti, generování klíče pro následující blok dat

1.2.1 Operace blokových šifer

Operace, jak již bylo uvedeno, je jedním z prvků, kterým se od sebe šifry liší. Operace jsou prováděny buď nad celým blokem nebo jeho částí. Nejpoužívanější operace jsou uvedeny níže.

1.2.1.1 Operace XOR

Operace XOR, neboli exkluzivní OR (nebo), nebo také exkluzivní disjunkce je využívána díky její rychlosti a jednoduchosti. Operace se provádí na základě následující tabulky.

Tabulka 1 -
Operace XOR

x	y	$x \oplus y$
0	0	0
0	1	1
1	0	1
1	1	0

Dva stejné bity vracejí hodnotu 0 a dva odlišné bity vracejí hodnotu 1. Pro značení operace se používá symbol plus v kolečku \oplus . Operace se řídí následujícím pravidlem.

$$x_i \oplus y_i = z_i \quad (1)$$

$$z_i \oplus y_i = x_i \quad (2)$$

- x_i – bit otevřeného textu na pozici i
- y_i – bit klíče na pozici i
- z_i – bit zašifrovaného textu na pozici i

Bezpečnost této operace spočívá zejména v tom, že pro každý zašifrovaný bit jsou dvě vstupní kombinace dvou bitů. Představme si zašifrovaný bit 0, tuto hodnotu můžeme získat dvěma způsoby $\rightarrow 0 \oplus 0$ nebo $1 \oplus 1$. Z toho důvodu je pro dešifrování potřebná znalost klíče, jinak je to pouhé hádání otevřeného textu. Operace se používá například u šifer DES, 3DES, AES, IDEA a dalších a taky při rozvoji klíče v režimech činnosti.

1.2.1.2 Operace sčítání modulo

Operace vrací zbytek po dělení součtu dvou čísel. Dělitel se obvykle nemění. Operace neslouží jen k zajištění bezpečnosti, ale zabezpečuje i to, že číselná hodnota nepřekročí stanovenou hranici. Například součet modulo 256 (2^8) nevrátí hodnotu větší jak 255, protože 256 modulo 256 (zbytek po dělení 256) je 0. Hodnota v rozmezí 0..255, která je reprezentována jedním bajtem, hraje významnou roli v informačních technologiích. Všechny znaky ASCII tabulky mají velikost jeden bajt. V mnoha programovacích jazycích má svůj vlastní datový typ – byte (bajt). Operaci sčítání modulo zapisujeme následovně.

$$(x_i + y_i) \bmod m = z_i \quad (3)$$

- x_i – bajt vstupních dat na pozici i
- y_i – bajt dat klíče na pozici i
- z_i – bajt výstupních dat na pozici i
- m – dělitel

Výše uvedený vztah se používá pouze pro šifrování. Operace sečtení modulo oproti operaci XOR používá pro dešifrování jiný vztah. Tzv. odečítání modulo.

$$(z_i - y_i) \bmod m = x_i \quad (4)$$

Platnost uvedeného vztahu můžeme dokázat následovným způsobem. Předpokládejme, že dělitel m bude 256. Takže mohou nastat dva případy

1. Součet hodnot x a y bude menší než 255 – zbytek po dělení z bude roven součtu hodnot x a y .
2. Součet hodnot x a y bude větší nebo rovno 255 \rightarrow zbytek po dělení z bude menší než x a zároveň menší než y a dále nebude větší jako 254.

To ovšem platí pouze za předpokladu, že

$$\forall x, y \in \langle 0, 255 \rangle$$

$$m = 256$$

Abychom dokázali platnost dešifrovací rovnice pro oba případy, musíme dokázat platnost pro každý zvlášť. Rozeberme si první případ. Pokud součet není větší jako 256, tak zbytek po dělení 256 je opět součet levé strany. Což znamená, že modulo v rovnici nehraje žádnou roli. Všechny hodnoty na pravé straně rovnice jsou menší než 256, modulo hodnoty rovnice nemění. Z toho důvodu modulo můžeme rovnice vypustit a dostáváme následující vztah.

$$x + y = z \tag{5}$$

Pokud známe hodnotu z (hodnota bajtu zašifrovaných dat) a hodnotu y (hodnota klíče), tak dostáváme rovnici o jedné neznámé a vypočteme hodnot x (hodnota bajtu nezašifrovaných dat).

$$z - y = x \tag{6}$$

Jak už bylo zmíněno výše, tak modulo hodnoty x , y a z nemění, takže ho můžeme dosadit na jakékoliv místo v rovnici. Například za rozdíl hodnot z a y a dostaneme námi požadovaný tvar pro dešifrování.

$$(z - y) \bmod 256 = x \tag{7}$$

Druhý případ je poněkud složitější. Pokud budeme opět předpokládat, že hodnoty x a y budou v intervalu 0..255, tak maximální zbytek po dělení součtu čísel x a y bude mít hodnotu 254. Pokud zkusíme dosadit hodnoty y a z do předpokládané rovnice pro dešifrování, dostaneme následující vztah

$$(z - y) \bmod 256 = x \rightarrow (254 - 255) \bmod 256 = x \rightarrow (-1) \bmod 256 = x \tag{8}$$

Modulo ze záporného čísla se vypočítá přičtením dělitele k zápornému číslu, což vyplývá z věty o kongruenci. Věta nám říká, že zbytky po dělení a , b číslem m jsou kongruentní, pokud se rovnají. Potom můžeme říci, že následující vztah platí.

$$a = b + k \cdot m \quad (9)$$

- a – první zbytek po dělení
- b – druhý zbytek po dělení
- m – dělitel
- k – libovolné celé číslo

Pokud známe hodnotu čísla b a hodnotu dělitele m můžeme z výše uvedeného vztahu spočítat všechny kongruentní čísla a ke zbytku b . [6] Například pro náš zbytek po dělení $b = -1$ číslem 256 platí tato rovnice.

$$a = -1 + k \cdot 256 \quad (10)$$

Pokud dosadíme za k hodnotu 1, dostáváme rovnici

$$a = -1 + 1 \cdot 256 \rightarrow a = 255 \quad (11)$$

Nyní se vraťme k potvrzení druhého případu, kdy zbytek po dělení dvou čísel x a y je menší než obě dvě hodnoty x a y , tak z toho plyne, že vždycky z rovnice pro dešifrování dostaneme rovnici, v které počítáme modulo ze záporného čísla. Rozdíl hodnot z a y bude záporný. Rovnici po dešifrování proto můžeme upravit následovně.

$$(z - y) \bmod 256 = x \rightarrow z - y + 256 = x \rightarrow x + y - 256 = z \quad (12)$$

Pokud předpokládáme krajní situaci, kdy hodnoty x i y budou rovny 255, tak dostaneme číslo 510, které má zbytek po dělení číslem 256 stejný, jako kdybychom číslo 256 od čísla 510 odečetli, proto přepíšeme modulo v rovnici následovně.

$$(x + y) - 256 = z \quad (13)$$

Dvě výše uvedené rovnice jsou ekvivalentní, proto můžeme považovat rovnici pro dešifrování v obou případech za dokázanou. Druhou metodou testování platnosti dešifrovací rovnice je napsání programu, který otestuje všechny kombinace čísel x , y na intervalu 0..255 a následně pro všechny výsledné hodnoty z spolu s odpovídajícími hodnotami y , otestuje platnost dešifrovací rovnice. Program pro ověření platnosti dešifrovací rovnice je napsaný v jazyce Python verze 3.x a zdrojový kód vypadá následovně.

```
def OverPlatnost():
    pocetFalse = 0
    for x in range(0,256):
        for y in range(0,256):
            z = (x + y) % 256
            if (z - y) % 256 != x:
                pocetFalse += 1
    if pocetFalse == 0:
        return True
    else:
        return False
```

Obrázek 3 – Zdrojový kód programu v jazyce Python verze 3.x pro ověření dešifrovací rovnice

```
>>> OverPlatnost()
True
```

Obrázek 4 - Výsledek běhu funkce pro ověření dešifrovací rovnice

Rovnice platí, pokud je návratová hodnota funkce OverPlatnost() True. Funkce vrací True, pokud neexistuje kombinace čísel x a y , pro kterou by dešifrovací rovnice neplatila. Ověření chování operace sčítání modulo byla věnována velká část, neboť je tato operace v navržené šifře využívána.

Bezpečnost této operace zajišťuje fakt, že bez znalosti klíče je nemožné určit hodnotu bajtu otevřeného textu z hodnoty bajtu šifrovaného textu. Například, uvažujme hodnotu bajtu šifrovaných dat 114. Bez znalosti klíče může být hodnota bajtu otevřeného textu jakákoliv z hodnot 0..255, protože ke každé hodnotě bajtu otevřeného textu existuje hodnota bajtu klíče tak, že zbytek po dělení z jejich součtu nám vrátí hodnotu 114. Přitom se obě dvě hodnoty x a y mohou měnit. Tudíž je takřka nemožné, pouze ze znalosti hodnoty bajtu šifrovaných dat, určit hodnot bajtu klíče. Dvojice (x,y) , které budou odpovídající pro hodnotu z bajtu šifrovaných dat, generuje následující vztah.

$$\sum_i (i, z - i), \forall i \in \langle 0, z \rangle, \quad (14)$$

$$\sum_i (i, (z - i) + 256), \forall i \in \langle z + 1, 255 \rangle \quad (15)$$

Počet dvojic (x,y) , které jsou řešením rovnice $(x + y) \bmod 256 = z$, se znalostí hodnoty z , je 256. Protože i může nabývat 256 různých hodnot na intervalu 0..255. Dále bude důkaz bezpečnosti podrobněji popsán v kapitole 5.1.6.

Operaci používá například šifra IDEA. A jak již bylo zmíněno, bude základem navržené šifry diplomové práce.

1.2.1.3 Operace násobení modulo

Operace násobení modulo je obdobná jako operace sčítání modulo s rozdílem, že vrací zbytek po dělení součinu dvou čísel. Operaci používá například šifra IDEA. Budeme předpokládat, že klíč a blok dat mají stejné délky a že se blok skládá s bajtů. Rovnice pro tuto operaci vypadá následovně.

$$(x_i \cdot y_i) \bmod m = z_i \quad (16)$$

- x_i – bajt dat otevřeného textu na pozici i
- y_i – bajt dat klíče na pozici i
- z_i – bajt zašifrovaných dat na pozici i
- m – dělitel

Dále se této operaci věnovat nebudeme.

1.2.1.4 Substituce a transpozice

Jedná se o jedny z nejstarších technik šifrování, které ale stále patří mezi nejpoužívanější techniky. Dokonce jsou jednou z nejdůležitějších částí dnešních blokových šifer. Nejedná se ovšem o jednoduchou (monoalfabetickou) substituci, která znaky nahrazuje z jedné abecedy, ale o polyalfabetické substituce, kdy abeced je mnohem víc. Používá je například DES a 3DES, AES a další.

Operace substituce spočívá v tom, že se jeden znak (bajt) otevřeného textu nahradí jiným znakem (bajtem) šifrovaného textu. [1] Nahrazení se obvykle provádí na základě klíče spolu se substituční tabulkou. Prvkům, které provádějí substituci u šifer DES, 3DES a AES, se říká tzv. S-Boxy. Substituce slouží k odstranění vztahu mezi klíčem a textem. Substituční tabulka se generuje pouze jednou, ještě před šifrováním. Takový příkladem Rijndael S-Box šifry AES. [5]

Transpozice znaky otevřeného textu nemění, ale podle klíče dochází k záměně pořadí znaků (bajtů) otevřeného textu. [1] V dnešní době se tato operace provádí matematic-

kou operací permutace, od toho P-Boxy šifry DES a 3DES. [5] U šifry AES, které se budeme věnovat samostatně později, jsou to operace

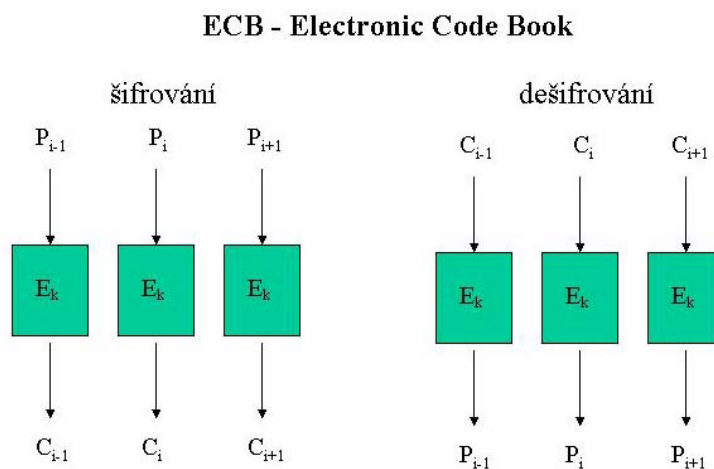
- ShiftRow Transformation, která posunuje pořadí znaků (bajtů) v tabulce nezávisle na klíči
- MixColumn Transformation, která mění pořadí znaků (bajtů) po sloupcích v závislosti na klíči, provádí se násobením polynomem [5]

1.2.2 Režimy činnosti – správa klíče (Key-Management)

Režim činnosti je u blokových šifer důležitým prvkem. Jedná se o činnost, při které dochází k rozšiřování (změně) klíče pro následující blok. Obvykle se odvozuje na základě kombinace - klíče, bloku otevřených dat a bloku šifrovaných dat. Pokud by ke změně klíče nedocházelo, bylo by to velkou slabinou blokových šifer. Neměnila by se pravidla pro šifrování všech bloků. Pokud by se všechny bloky šifrovaly stejně, dalo by se jednodušeji získat klíč pro šifrování. [5] V následujících částech si přiblížíme princip čtyř základních režimů činnosti, nicméně jich existuje mnohem víc. Vstupy a výstupy se dají provázat více způsoby.

1.2.2.1 Režim činnosti ECB – Electronic Code Book

Tento režim není režimem v pravém slova smyslu, neboť nedochází k žádným změnám klíče. Pro všechny bloky je používán stejný klíč. [5]

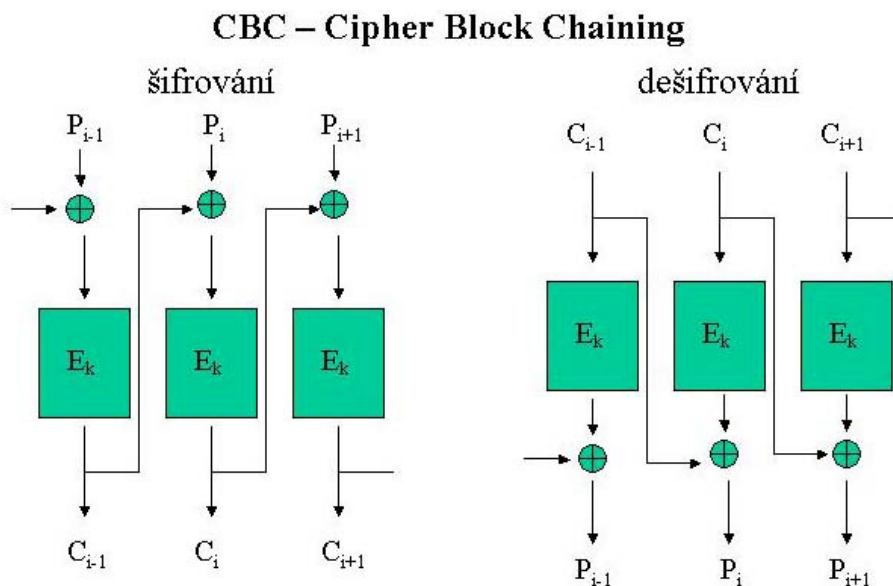


Obrázek 5 - Režim činnosti ECB – Electronic Code Book [5]

- P_i – blok nezašifrovaných dat
- E_k – šifrovací/dešifrovací algoritmus
- C_i – blok zašifrovaných dat

1.2.2.2 Režim činnosti CBC – Cipher Block Chaining

Pro první blok se klíč odvozuje operací XOR, kdy je sečten inicializační vektor s blokem nezašifrovaných dat. Pro další bloky je sečten blok zašifrovaných dat spolu s dalším blokem nezašifrovaných dat. [5] Schéma činnosti je na následujícím obrázku.



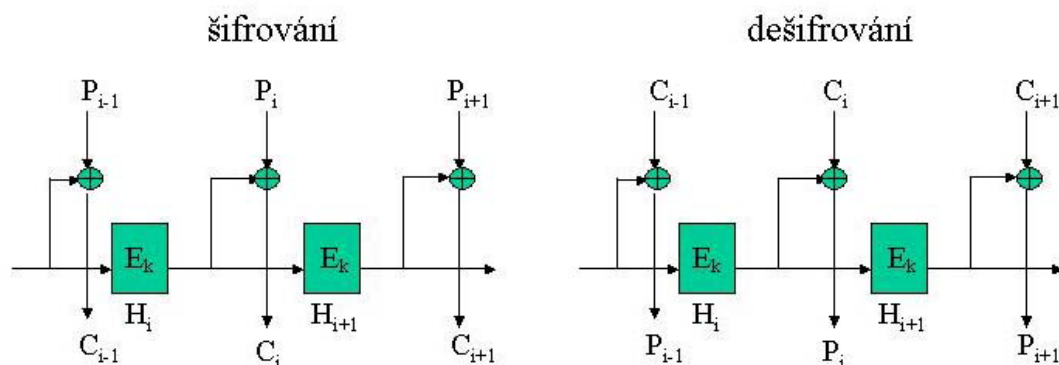
Obrázek 6 - Režim činnosti CBC - Cipher Block Chaining [5]

- P_i – blok nezašifrovaných dat
- E_k – šifrovací/dešifrovací algoritmus
- C_i – blok zašifrovaných dat

1.2.2.3 Režim činnosti OFB – Output Feedback Block

Tento režim se označuje jako simulace proudové šifry, protože na bloky nezašifrovaných dat nejsou aplikované operace pro šifrování. Operace XOR probíhá mezi blokem vstupních nezašifrovaných dat a výstupem blokové šifry. Bloková šifra zde slouží pouze jako pseudonáhodný generátor posloupnosti. První blok vstupních dat je přičten operací XOR k inicializačnímu vektoru. [5] Schéma činnosti je vyobrazeno na následujícím obrázku.

OFB – Output Feedback Block



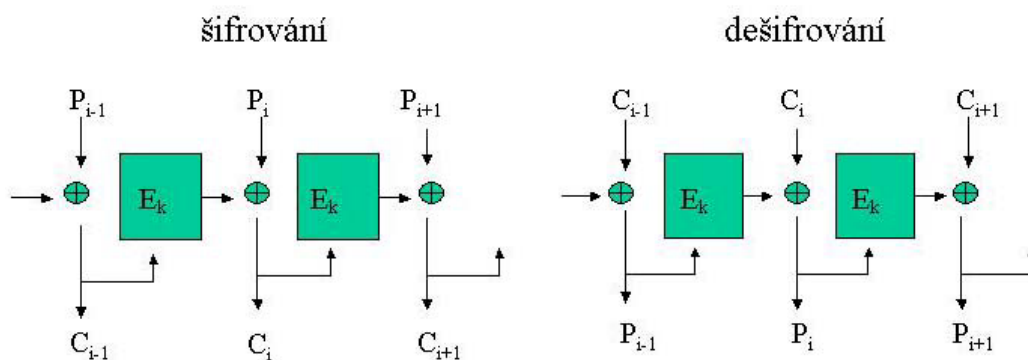
Obrázek 7 - Režim činnosti OFB - Output Feedback Block [5]

- P_i – blok nezašifrovaných dat
- E_k – šifrovací/dešifrovací algoritmus
- C_i – blok zašifrovaných dat po operaci XOR
- H_i – hodnota pseudonáhodné posloupnosti

1.2.2.4 Režim činnosti CFB – Cipher Feedback Block

Funguje na obdobném principu jako režim OFB – Output Feedback Block. Data nejsou šifrovány pomocí blokových operací šifrovacího algoritmu. Šifrování probíhá za využití operace XOR, kdy je ke vstupním datům přičten inicializační vektor. Jako další klíč je používán výstup první operace XOR, který je upraven šifrovacím algoritmem. [5] Princip činnosti opět znázorňuje následující obrázek.

CFB – Cipher Feedback Block

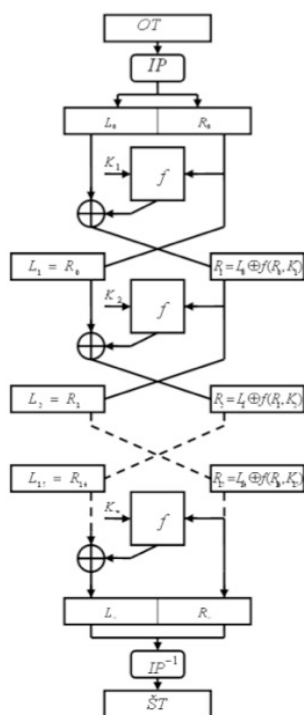


Obrázek 8 - Režim činnosti CFB - Cipher Feedback Block [5]

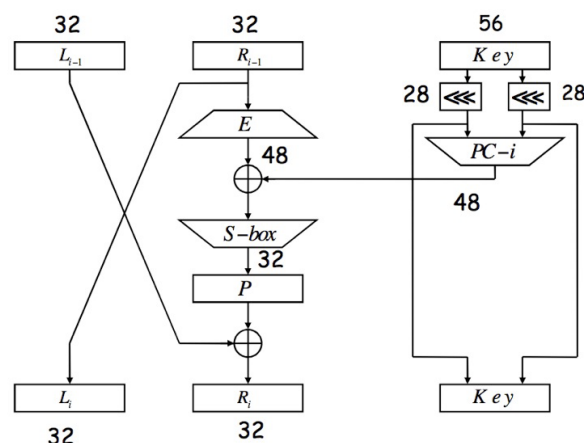
- P_i – blok nezašifrovaných dat
- E_k – šifrovací/dešifrovací algoritmus
- C_i – blok zašifrovaných dat po operaci XOR

1.2.3 Bloková šifra DES

Jedná se o jednu z prvních blokových šifer. Šifruje bloky o délce 64 bitů. K šifrování se používá klíč o délce 64 bitů, ale platných pro šifrování je pouze 56 bitů, neboť každý osmý bit je paritní a algoritmus jej ignoruje. Operacemi pro šifrování jsou pouze S-Boxy (substituční metoda) a P-Boxy (transpoziční metoda). Operace jsou aplikovány 16 krát → celkem 16 rund. Schéma činnosti vypadá následovně. [5]



Obrázek 9 - Blokové schéma šifry DES [5]

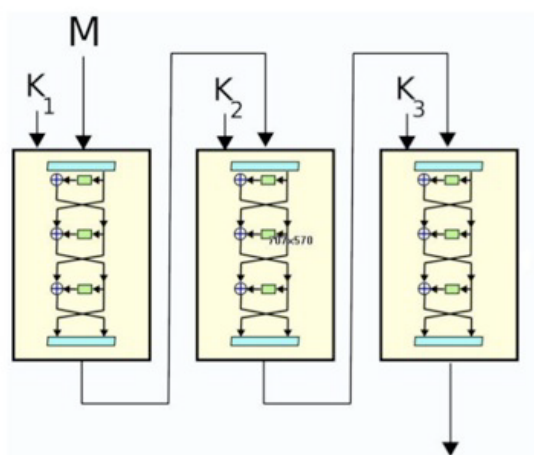


Obrázek 10 - Schéma jedné rundy [5]

Bohužel, díky délce klíče 56 bitů, byla šifra roku 1997 prolomena hrubou silou za využití 78 tisíc počítačů. A v roce 1998 byl sestrojen počítač s názvem DES Cracker, který byl schopen spočítat DES klíč za 5 dnů.

1.2.4 Bloková šifra 3DES (Triple-DES)

Tato bloková šifra vznikla v roce 1998 v reakci na prolomení šifry DES. Jednalo se o trojnásobné aplikování DESu s třemi klíči o délce 56 bitů. Takže šifra má délku klíče solidních 168 bitů. [5] Princip průběhu šifrování vypadá následovně.



Obrázek 11 - Průběh šifry 3DES [5]

Existuje mnoho variant šifry 3DES lišící se v aplikaci a chodu šifry. Dále v textu jsou uvedeny vybrané varianty.

- DES EEE3 – uveden na obrázku výše.
- DES EDE3 – zašifrování prvním klíčem, dešifrování druhým klíčem a zašifrování třetím klíčem. Dešifrování by probíhalo inverzně ve tvaru DED3 – dešifrování třetím klíčem, zašifrování druhým klíčem a dešifrování prvním klíčem.
- DES EEE2 – šifrování prvním klíčem, šifrování druhým klíčem a šifrování třetím klíčem.
- DES EDE2 – šifrování prvním klíčem, dešifrování druhým klíčem a šifrování prvním klíčem. Jedná se o nejpoužívanější formu.
- A další. [5]

Tato šifra nebyla dosud prolomena. [5]

1.2.5 Bloková šifra AES

Opět se jedná o šifru, která vznikla v reakci na prolomení šifry DES v roce 1998. Zkratka AES znamená Advanced Encryption Standard. Šifruje bloky dat o velikosti 128 bitů (volitelně 192 a 256). Podle toho je dlouhý i klíč – 128, 192 a 256 bitů a počet rund –

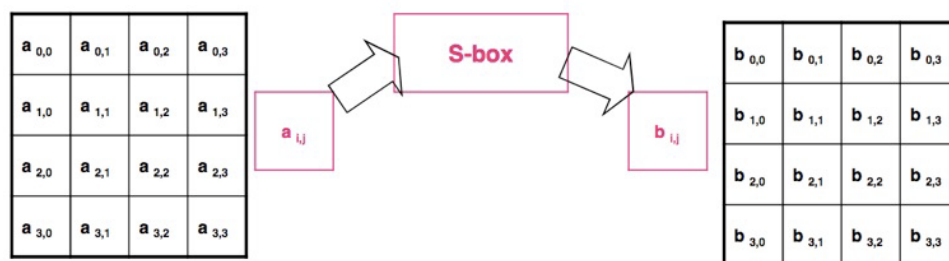
10, 12 a 14. Šifrování probíhá rychlostí, která je větší jak 45 MB/s. [5] Počet možných klíčů vyplývá z následující rovnice

$$\text{Počet klíčů} = 2^{\text{délka klíče}} \quad (17)$$

V dnešní době je výpočetně možné hrubou silou prolomit klíč délky přibližné 80-bitů. Teoreticky za použití kvantových počítačů dosáhneme prolomení klíče délky 160. Takže můžeme považovat varianty AES-192 a AES-256 v dnešní době a „blízké“ budoucnosti za bezpečné a neprolomitelné. Pokud se nenajde jiná slabina nebo algoritmus pro prolomení. [5]

AES používá následující operace:

- ByteSub Transformation – Jedná se o substituci na bajtové úrovni, která je nelineární (nelze vyjádřit rovnicí) vrstvou a tím zvyšuje odolnost vůči lineární a diferenciální kryptoanalýze. [5]

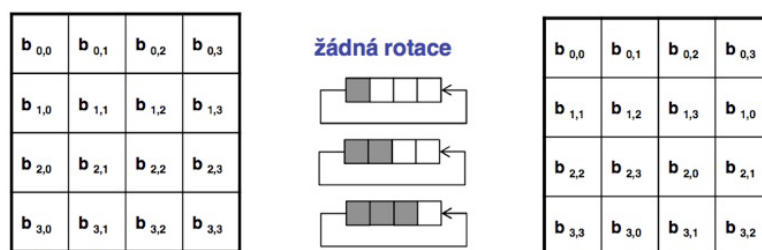


Obrázek 12 - AES - ByteSub Transformation [5]

The diagram shows the internal structure of the S-Box. It is represented as a matrix equation:
$$\begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$
The output vector y is shown on the left. A red arrow labeled a_{ij}^{-1} points to the matrix. A red arrow labeled 'S-box' points to the entire equation.

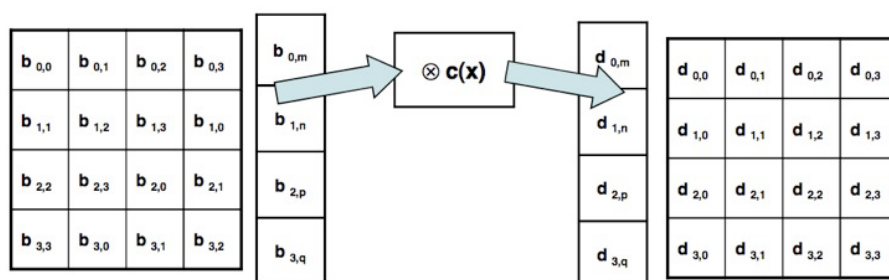
Obrázek 13 - Struktura S-Boxu [5]

- ShiftRow Transformation – Lineární vrstva pro posun bajtů v tabulce



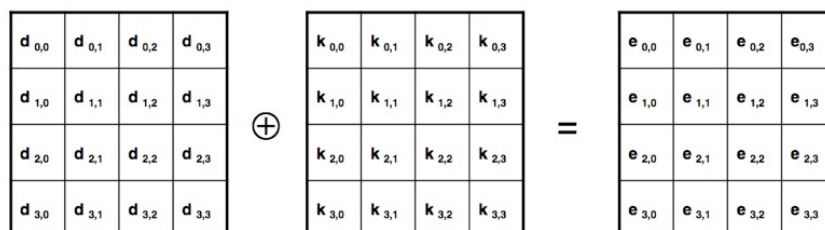
Obrázek 14 - AES - ShiftRow Transformation [5]

- MixColumn Transformation – Mixování buněk (bajtů) ve sloupcích tabulky (sloupce se násobí polynomem a implantováno pomocí operace XOR) [5]



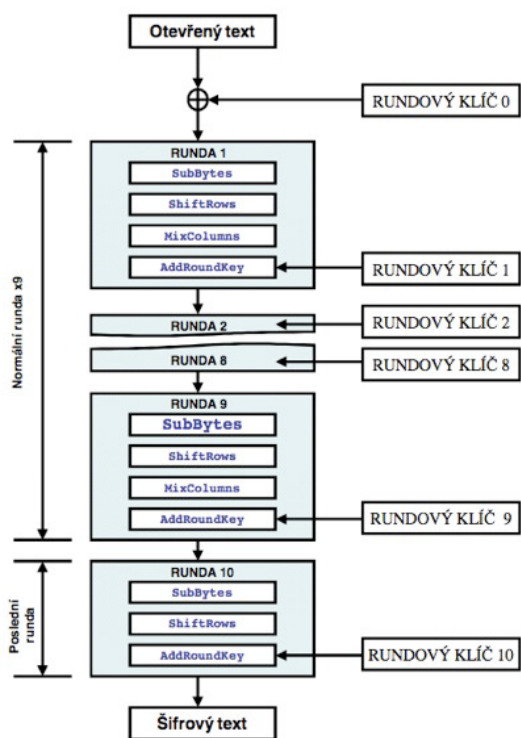
Obrázek 15 - AES - MixColumn Transformation [5]

- Add Round Key – Přičtení rundového klíče k datům operací XOR. [5]

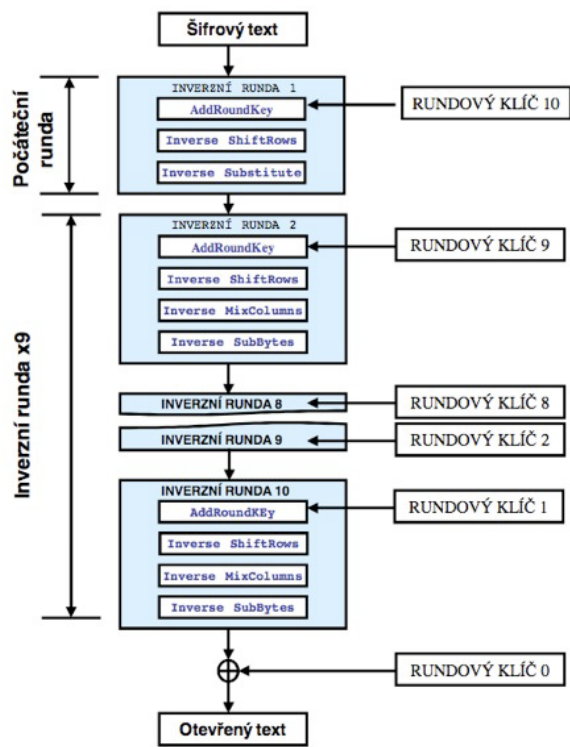


Obrázek 16 - AES - Add Round Key [5]

Jak už bylo zmíněno, celý průběh probíhá v 10, 12 a 14 rundách. V každé rundě jsou prováděny 4 výše uvedené operace, vyjma poslední, která neobsahuje MixColumn Transformation, protože je výpočetně nejnáročnější a i bez ní je bezpečnost zaručená. Celý průběh je znázorněn na dalším obrázku. [5]



Obrázek 17 - AES – šifrování [5]



Obrázek 18 - AES – dešifrování [5]

2 NÁVRH SYMETRICKÉ BLOKOVÉ ŠIFRY TEORETICKY

V této části se budeme věnovat teoretickým předpokladům, jak postupovat při návrhu nové symetrické blokové šifry. Popíšeme si problémy, které bude nutné vyřešit a které prvky by měla navržená šifra obsahovat.

Nejprve je nutné se zamyslet nad otázkou, jestli se tvorba nové symetrické blokové šifry v dnešní době vyplatí. Šifer je v dnešní době velké množství. Navíc existuje šifra AES-256, která se považuje i do budoucna za neprolomitelnou. Za předpokladu, že nikdy nebude AES-256 prolomen může odpověď znít v neprospěch tvorby nové šifry. Mohlo by se jednat o plýtvání času. Na druhou stranu je AES neustále testován a jsou podnikány pokusy o prolomení, že je jen otázka času, kdy k tomu dojde. Další argument, proč žádnou novou šifru nenavrhopat je, že AES je velmi rychlý s velkým stupněm bezpečnosti. V dnešní době poměrem rychlost/bezpečnost nemá konkurenci. Důvody k tvorbě nové šifry jsou následující.

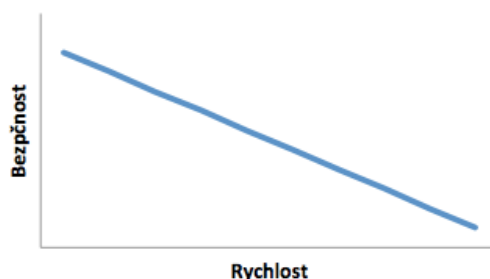
1. Z důvodu hypotetického prolomení AESu, je dobré si držet „zadní vrátka“, jelikož se AES využívá čím dál víc a prolomení by mohlo vést ke katastrofickým následkům.
2. Čím více se bude používat šifer, tím se síly kryptoanalytiků rozprostřou na více cílů. Prolomení se může prodloužit.
3. Dále, čím více bude používaných šifer, tím obtížnější bude pro útočníka určit, kterou šifrou byla data zašifrována.
4. Dalším důvodem může být objevování nových možností šifrování, pokud se tvorbě nových šifer budeme věnovat.

Po krátké úvaze, se nyní budeme věnovat části, kde si rozebereme otázky spojené s návrhem symetrické blokové šifry. Dále budou popsány části, bez kterých se bloková šifra neobejde, a které bude nutné navrhnout. Jelikož je náplní diplomové práce návrh blokové šifry, tak všechny informace budou koncipovány s ohledem na návrh blokové šifry. Pro jiné druhy šifer bychom museli použít jiné prvky, pravidla a předpoklady.

2.1 Bezpečnost nebo rychlost

Jedna ze zásadních otázek je, jestli chceme, aby navržená šifra byla rychlá nebo bezpečná. Ideální šifra by byla nejrychlejší a nejbezpečnější, nicméně taková dosud neexistuje. Situaci popisuje následující výrok Vincenta Rijmena, který řekl: „Security always

comes at a cost to performance.“. Výrok můžeme přeložit následovně – „Bezpečnost je vždy za cenu výkonu.“. Z čehož plyne, že platí nepřímá úměra mezi bezpečností a rychlostí. Čím bezpečnější bude šifra navržená, tím bude pomalejší.



Obrázek 19 – Aproximace vztahu mezi bezpečností a rychlostí šifry

Samozřejmě za předpokladu, že bude šifra navržena správně. Uvedu příklad na monoalfabetické Caesarově šifře. I kdybychom text zašifrovali pomocí Caesarovy šifry milionkrát, tak bude navýšení bezpečnosti zanedbatelné, ne-li nulové.

Rychlost je i hodně ovlivněna implementací šifry a implementací operací šifry. Proto může být navržená šifra v teoretické rovině mnohem rychlejší, než potom ve výsledku.

Je nutné si vybrat, zda budeme šifru navrhovat v závislosti na bezpečnosti nebo spíše na rychlosti, neboť toto rozhodnutí nám v největší míře ovlivní následující postup. Zejména návrh délky klíče, počet rund, operace, režim činnosti (správu klíče), v podstatě vše ostatní.

2.2 Délka klíče

Další důležitá věc, které se musíme věnovat při návrhu nové šifry je beze sporu délka klíče. Pokud je šifra dobře navržena a jedinou šancí na prolomení bude útok hrubou silou na klíč, tak platí následující pravidlo. Čím delší klíč, tím je čas potřebný na vyzkoušení všech klíčů hrubou silou větší. Neboli, pokud chceme bezpečnější šifru, zvolíme délku klíče větší. Vzorec na výpočet počtu možných klíčů byl uveden 1.2.5 *Bloková šifra AES* a vypadá následovně.

$$\text{Počet klíčů} = 2^{\text{délka klíče}} \quad (18)$$

Jak již bylo uvedeno v prvním odstavci kapitoly 1.2.5 *Bloková šifra AES*, není možné v dnešní době za použití současných technologií v rozumném čase otestovat všechny klíče o délce 81 bitů a víc. Dokonce na otestování klíče o délce 256 bitů bychom potřebovali tolik energie, jako je energie výbuchu supernovy. [5] Tato hodnota vyplývá z energie potřebné na změnu stavu z 0 na 1. Nicméně není jasné, jak to bude v budoucnosti, proto bych se nebál použít délku klíče i větší jak 256 bitů. Příkladem je například šifra Whirpool, která je prakticky až na malé změny šifra AES s délkou klíče 512 bitů.

2.3 Operace pro šifrování/dešifrování

Operace prováděné nad bloky dat jsou základním stavebním kamenem všech blokových šifer. Volba operací ovlivňuje celou šifru závratným způsobem jak po stránce rychlosti, tak po stránce bezpečnosti. Nejčastěji používané operace byly popsány a rozebrány výše v kapitole 1.2.1 *Operace blokových šifer*. Většinou si nevystačíme s jednou operací, ale je nutné použít kombinaci více operací, které stíží prolomení šifry. Jako příklad lze uvést téměř všechny dnes používané blokové šifry. DES – 2 operace plus operace XOR, AES – 4 operace plus operace XOR, IDEA – 3 operace včetně operace XOR, a tak dále. Zde nejvíce platí výše zmíněný výrok Vincenta Rijmena. Neboť čím více operací použijeme, tím bude šifra hůře prolomitelná, nicméně bude o to pomalejší. V praxi se volí kompromis: šifra musí být bezpečná, ale ne moc pomalá. Pokud bereme v úvahu rozvoj technologií včetně počítačů, nebude v budoucnu nutné si dělat starosti s počtem a druhem operací. Samozřejmě rostoucí výkon nemusí znamenat klesající bezpečnost, neboť pokud bude jedinou možností útoku na šifru útok hrubou silou, tak bezpečnost nebude mít nic společného s použitými operacemi. Pro dobře navrženou šifru s dobře navrženými operacemi znamená pokrok nárůst rychlosti a ne pokles bezpečnosti. Pokud bychom se rozhodli šetřit na operacích, abychom docílili výkonu v rychlosti, tak je jen otázkou času, kdy bude objevena metoda k prolomení šifry jinou cestou.

Použité operace by měly být odolné vůči všem známým kryptoanalytickým technikám včetně diferenciální a lineární kryptoanalýzy. Odolnost vůči diferenciální kryptoanalýze můžeme zajistit operacemi – sečtení modulo, násobení modulo, XOR. Odolnost vůči lineární kryptoanalýze nám zajišťuje operace substituce, která má nelineární chování. Šifry by též měly mít inverzní charakter, aby se daly dešifrovat.

V souhrnu můžeme říct, že návrh operací jedna z nejdůležitějších částí návrhu nové šifry, protože ovlivňuje rychlost i bezpečnost. Základní kombinací, která se používá, je právě různě modifikovaná substituce a transpozice spolu s operací XOR.

2.3.1 Velikost bloku

Při návrhu nové blokové šifry je nutné si rozmyslet i velikost bloku. Obvykle se používá stejná délka bloku, jako je délka klíče. To ovšem neplatí u osobité šifry IDEA, která k šifrování bloků o délce 64 bitů, používá klíč o délce 128 bitů. [5]

2.3.2 Režim činnosti – správa klíče (Key-Management)

Další zásadní prvek nové blokové šifry je návrh režimu činnosti. Jak už bylo zmíněno, pokud bychom režim činnosti vynechali nebo by se jednalo o režim ECB – Electronic Code Book, došlo by k velkému snížení bezpečnosti navržené šifry. Při návrhu se nemusíme omezit pouze na 4 zmíněné režimy v kapitole 1.2.2 *Režimy činnosti*, jelikož je jich mnohem více. Při tvorbě klíče pro následující blok můžeme kombinovat aktuální klíč s aktuálně zašifrovanými daty, klíč se vstupními daty, vstupní data, klíč i výstupní zašifrovaná data a nebo můžeme vytvořit svůj vlastní. Dle mého názoru je nutné tento prvek do šifry začlenit a rozhodně nepoužívat variant ECB.

Nicméně každý prvek, tedy i režim činnosti, zpomaluje průběh šifrování, ale dělá šifru bezpečnější.

2.4 Struktura šifrování/dešifrování a počet rund

Při návrhu nové šifry musíme navrhnout i její strukturu, tzn. sled operací a počet rund (počet kol opakování operací). Struktura zobrazuje obvykle jednu rundu, která se v závislosti na počtu rund opakuje. Znázorňuje též rozložení operací, vstupy pro rundové klíče (klíče použité jen v jedné rundě) a další operace, které se provádějí v rámci jedné rundy. Dešifrování obvykle probíhá opačným směrem než šifrování a to ze zdola nahoru za využití inverzních operací k operacím pro šifrování.

2.5 Tvorba inicializačního vektoru

Pokud se rozhodneme používat režim činnosti, tak se při návrhu budeme muset věnovat i tvorbě inicializačního vektoru. Například jeho odvození od uživatelského hesla. Pro inicializační vektor by mělo platit, aby z jeho znalosti nešlo zpětně heslo určit nebo

odvodit. Tuto roli nám dobře zajistí například hashovací funkce (například SHA-256, která je brána za bezpečnou), které se vyznačují schopností jednosměrnosti. Hashovací funkce mají i tu výhodu, že nám vrací hodnotu o pevné délce pro jakýkoliv vstup. Takže se nemusíme starat o doplňování na patřičnou délku klíče.

2.6 Doplnění necelého bloku

Blokové šifry pracují s bloky dat o pevné délce, nicméně mohou nastat případy, kdy je potřeba pracovat s blokem menším, než je velikost bloku. Pokud se šifrují data o velikosti menší jak velikost bloku nebo pokud není velikost dat násobkem velikosti bloku, je nutné v takovém případě doplnit data na velikost bloku. Existuje na to mnoho způsobů, jak blok doplnit. Například:

- Doplnění o nuly.
- ANSI X.923 – Na konec se doplní nuly a bajt udávající počet doplněných znaků (délku doplňku).
- ISO 10126 – Na konec se doplní náhodně generované znaky (bajty) a poslední bajt udává počet doplněných znaků.
- PKCS57 – Každý doplněný bajt má hodnotu počtu doplněných bajtů, takže pokud budeme potřebovat doplnit 8 bajtů, tak blok doplníme o 8 bajtů hodnoty 8. [6]

Záleží pouze na nás, který princip zvolíme, nebo jestli si vytvoříme vlastní. Ta to část nám chování šifry a její bezpečnost s rychlostí moc neovlivní, nicméně se bez ní neobejdeme.

II. PRAKTICKÁ ČÁST

3 NÁVRH BLOKOVÉ ŠIFRY

Tato část diplomové práce je věnována vlastnímu návrhu blokové šifry. V první části obsahuje popis navržených prvků a vlastnosti blokové šifry. V další části bude popsán průběh šifrování a dešifrování souboru a v jednotlivých podkapitolách budou popsány prvky blokové šifry.

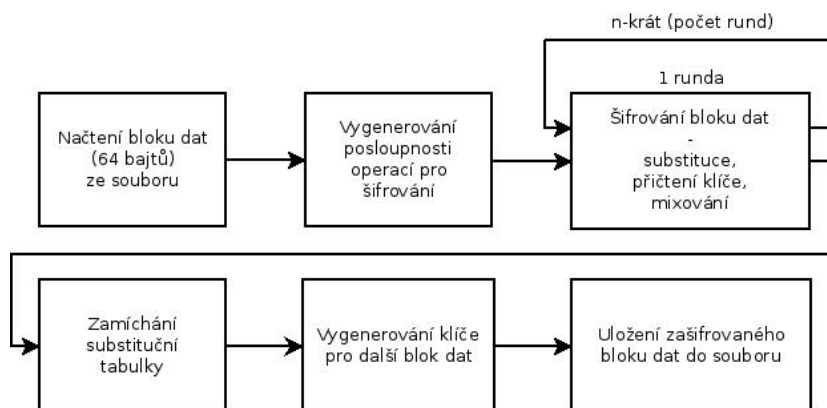
3.1 Základní prvky a vlastnosti navržené blokové šifry

Jak už vyplývá z názvu této diplomové práce, jedná se o návrh symetrické šifry. Pro návrh byla zvolena bloková šifra. Navržená bloková šifra má následující vlastnosti a prvky.

- Délka klíče – 512 bitů.
- Velikost bloku dat – stejná jako délka klíče - 64 bajtů (512 bitů).
- Počet rund – variabilní - volí uživatel (rozmezí 1 až 255) – zakódování klíčem.
- Generování inicializačního vektoru – na základě uživatelského hesla a pomocí hashovací funkce SHA-256. – inicializační vektor se neukládá.
- Uložení otisku inicializačního vektoru hashovací funkcí SHA-256 do výstupního souboru. – Autorizace uživatele při dešifrování.
- Operace zamíchání substituční tabulky – pomocí klíče.
- Operace jedné rundy – substituce, přičtení klíče k datům a mixování (transpozice).
- Pořadí operací rundy – variabilní – generování pomocí klíče.
- Režim činnosti (generování klíče pro další blok) – variabilní – na základě klíče.
- Doplnění bloku o velikosti menší než 64 bajtů – náhodnými znaky a zakódování počtu doplněných znaků klíčem.
- Šifrování a dešifrování probíhá po blocích – není potřeba načíst celý soubor, ale načítá se vždy jeden blok – 64 bajtů a ten se šifruje nebo dešifruje. Zašifrované nebo dešifrované data jsou průběžně ukládána.
- Šifra je asynchronní – šifrování a dešifrování neprobíhá stejnou dobu.
- Doba šifrování se mění v závislosti na vstupních datech.

Průběh šifrování je následující. Pro šifrování si uživatel zvolí heslo, zadá jméno souboru pro šifrování a určí počet rund. Nejprve se z uživatelského hesla vygeneruje inicializační vektor a vypočítá se hash otisk inicializačního vektoru pro ověření uživatele u dešifrování. Poté se na základě inicializačního vektoru zakóduje bajt udávající počet rund, proto je možnost 1 až 255 rund. Zakódovaný počet rund plus hash otisk inicializačního

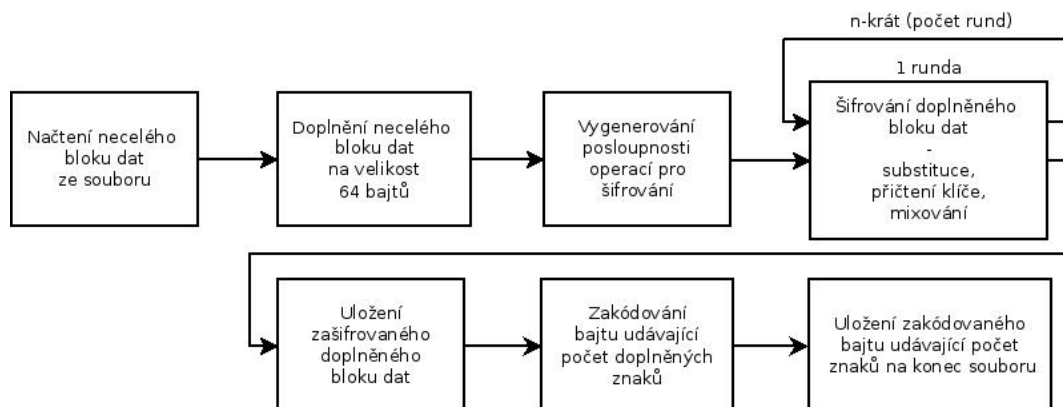
vektoru uloží na začátek zašifrovaného souboru. Na základě inicializačního vektoru dojde k zamíchání pevně dané vstupní substituční tabulky. Následně se zjistí počet bloků a případně počet bajtů necelého bloku dat. Probíhá šifrování celých bloků, kdy průběh šifrování jednoho bloku vypadá následovně:



Obrázek 20 - Blokové schéma šifrování celého bloku dat

1. Načtení bloku (64 bajtů) nezašifrovaných dat ze vstupního souboru.
2. Vygenerování posloupnosti operací na základě počtu rund a klíče (pro každou rundu je sled operací jiný).
3. Zašifrování bloku dat n -krát operacemi – substituce, přičtení klíče a mixování v závislosti na posloupnosti operací pro rundu (číslo n je počet rund).
4. Zamíchání substituční tabulky pomocí klíče.
5. Vygenerování klíče pro další blok dat.
6. Uložení zašifrovaného bloku dat se do výstupního souboru.

Jestli vstupní soubor není násobkem 64 bajtů, probíhá doplnění a zašifrování posledního necelého bloku dat. Pokud se jedná o soubor o velikosti menší než 64 bajtů, dochází k zašifrování prvního bloku dat a průběh je následovný:



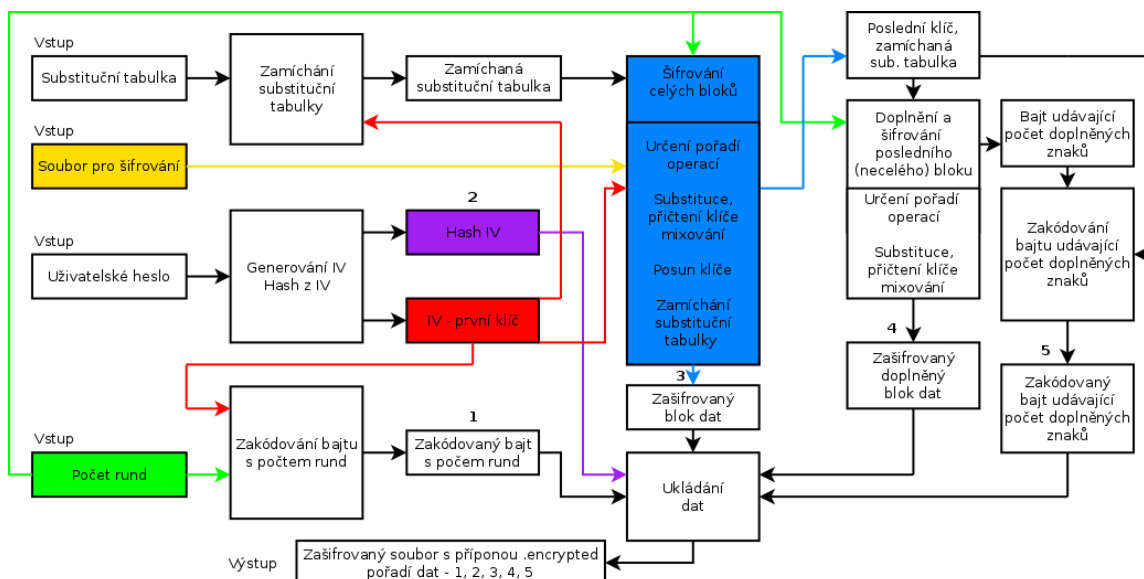
Obrázek 21 - Blokové schéma šifrování necelého bloku dat

1. Načtení necelého bloku nezašifrovaných dat ze vstupního souboru.
2. Doplnění bloku dat o i bajtech o 64- i náhodně generovaných znaků.
3. Vygenerování posloupnosti operací na základě počtu rund a klíče (pro každou rundu je sled operací jiný).
4. Zašifrování bloku dat n -krát operacemi – substituce, přičtení klíče a mixování v závislosti na posloupnosti operací pro rundu (číslo n je počet rund).
5. Uložení zašifrovaného bloku dat do výstupního souboru.
6. Zakódování bajtu udávající počet doplněných znaků pomocí klíče (IV – první blok).
7. Uložení zakódovaného bajtu udávající počet doplněných znaků na konec výstupního souboru.

Výstupem šifrování je zašifrovaný soubor obsahuje data v následujícím pořadí:

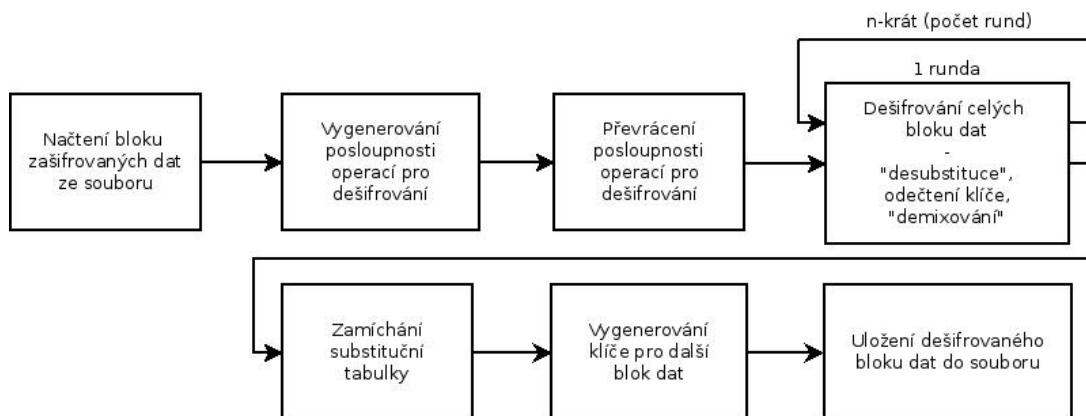
1. zakódovaný bajt udávající počet rund
2. hash inicializačního vektoru pro autorizaci uživatele při dešifrování
3. zašifrované bloky dat včetně necelého
4. zakódovaný bajt udávající počet doplněných znaků, pokud byl poslední blok necelý

Soubor je uložen s příponou .encrypted. Celý průběh šifrování zachycuje následující blokové schéma.



Obrázek 22 - Blokové schéma průběhu šifrování souboru

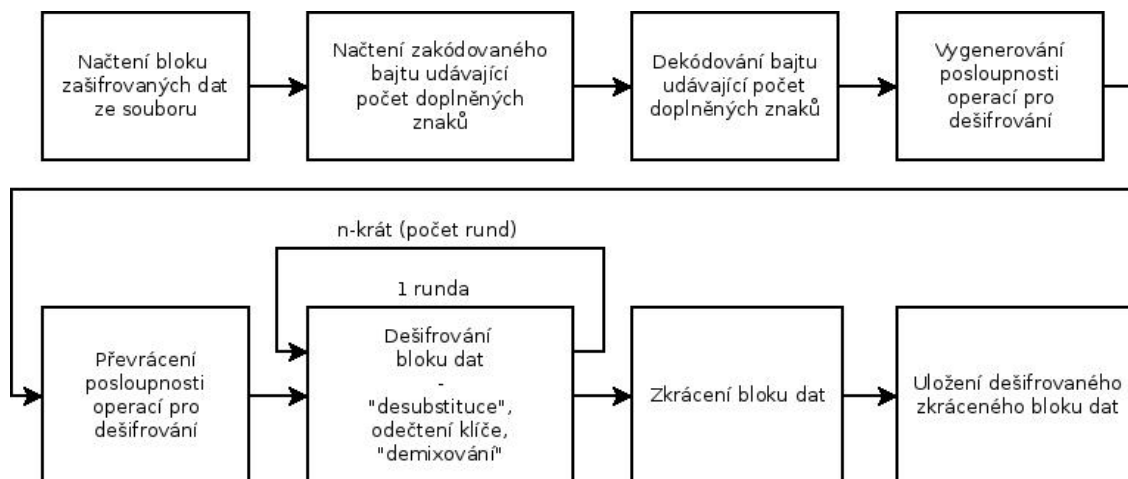
Nyní k průběhu dešifrování, které je obdobné jako průběh šifrování. Pro dešifrování je uživatelem zvolen zašifrovaný soubor s příponou `.encrypted`. Uživatel zadá heslo pro dešifrování, které předtím použil pro zašifrování souboru. Následně se načte zakódovaný počet rund a hash inicializačního vektoru. Na základě uživatelského hesla se vygeneruje inicializační vektor, stejně jako při šifrování, a vypočítá se hash otisk spočteného inicializačního vektoru. Vygenerovaný hash otisk inicializačního vektoru a načtený hash ze zašifrovaného souboru se porovnají, dojde k autorizaci uživatele. Pokud se hash otisky shodují, přistoupí se k dešifrování. Ale ještě před samotným dešifrováním jednotlivých bloků dat se pomocí inicializačního vektoru dekóduje počet rund a na základě inicializačního vektoru se zamíchá vstupní pevně daná substituční tabulka, která je stejná jako tabulka pro šifrování. Nyní se zjistí počet bloků (při dešifrování už jsou všechny bloky o velikosti 64 bajtů, protože při šifrování byl necelý blok dat doplněn na velikost 64 bajtů) a zda je či není poslední blok doplněný (soubor má buď velikost $n \cdot 64 + 1$ bajtů bez doplněného posledního bloku nebo $(n \cdot 64 + 2)$ bajtů, pokud byl poslední blok doplněný – n je počet zašifrovaných bloků) a následně dochází k jejich dešifrování. Dešifrování jednoho bloku zašifrovaných dat probíhá v následujících krocích:



Obrázek 23 - Blokové schéma dešifrování celého bloku dat

1. Načtení bloku (64 bajtů) zašifrovaných dat ze vstupního souboru.
2. Vygenerování posloupnosti operací na základě počtu rund a klíče (pro každou rundu je sled operací jiný).
3. Převrácení posloupnosti operací, protože je posloupnost operací generována stejně jako pro šifrování.
4. Dešifrování bloku dat n -krát operacemi, které jsou inverzní k operacím pro šifrování – „de-substituce“, odečtení klíče a „demixování“ v závislosti na převrácené posloupnosti operací pro rundu (číslo n je počet rund).
5. Zamíchání substituční tabulky pomocí klíče.
6. Vygenerování klíče pro dešifrování dalšího bloku.
7. Uložení dešifrovaného bloku dat do výstupního souboru.

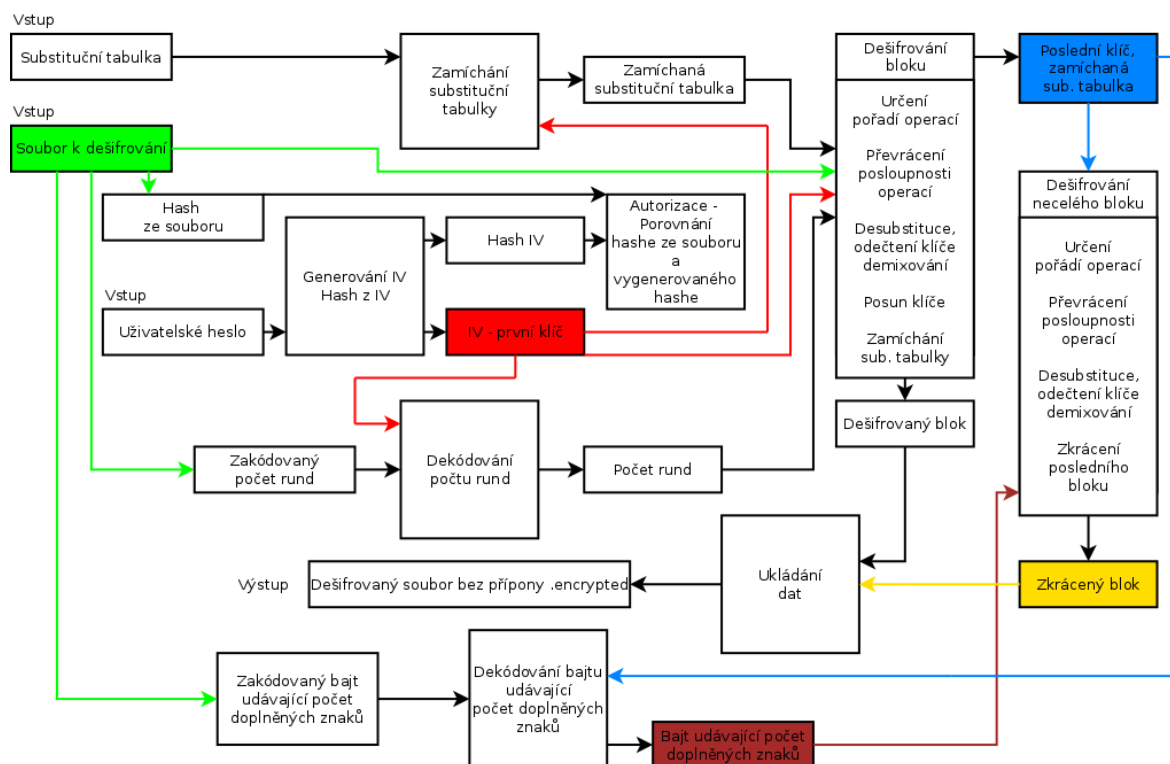
Tento proces se opakuje i -krát. Pokud byl poslední blok při šifrování doplněný, proces probíhá $(i-1)$ -krát a pro poslední doplněný blok je dešifrování následovné:



Obrázek 24 - Blokové schéma dešifrování necelého bloku dat

1. Načtení bloku (64 bajtů) zašifrovaných dat ze vstupního souboru.
2. Načtení posledního bajtu, který je zakódovaným bajtem udávající počet doplněných bajtů při šifrování.
3. Dekódování bajtu udávající počet doplněných znaků pomocí klíče.
4. Vygenerování posloupnosti operací na základě počtu rund a klíče (pro každou rundu je sled operací jiný).
5. Převrácení posloupnosti operací, protože je posloupnost operací generována stejně jako pro šifrování.
6. Dešifrování bloku dat n -krát operacemi, které jsou inverzní k operacím pro šifrování – „de-substituce“, odečtení klíče a „demixování“ v závislosti na převrácené posloupnosti operací pro rundu (číslo n je počet rund).
7. Zkrácení posledního dešifrovaného bloku na základě hodnoty bajtu udávající počet doplněných bajtů při šifrování.
8. Uložení dešifrovaného zkráceného blok na konec výstupního souboru.

Výsledkem dešifrování je dešifrovaný soubor s odebranou příponou .encrypted. Celý průběh dešifrování znázorňuje následující blokové schéma.

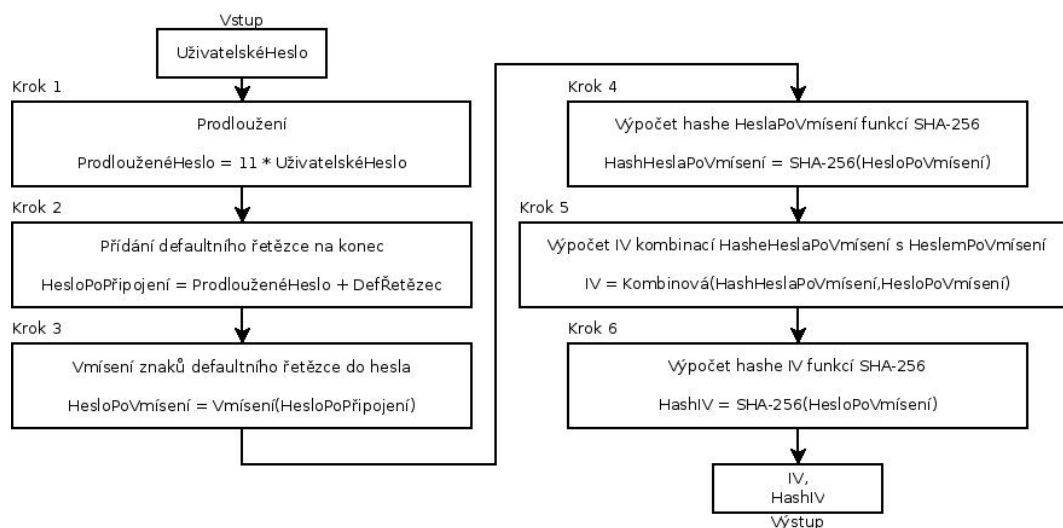


Obrázek 25 - Blokové schéma průběhu dešifrování souboru

V následujících kapitolách bude popsán princip činnosti jednotlivých prvků navržené šifry.

3.2 Generování inicializačního vektoru z uživatelského hesla a hashe inicializačního vektoru

Protože navržená šifra nevyužívá režim činnosti ECB – Electronic Code Book, popsaný v kapitole 1.2.2 věnované režimům činnosti, bylo nutné navrhnout generování inicializačního vektoru, dále jen *IV*. *IV* je generován z uživatelem zadaného hesla pro šifrování/dešifrování. Na konci je pro autorizaci uživatele při dešifrování vypočten hash *IV*. Celý průběh ukazuje následující blokové schéma.



Obrázek 26 - Blokové schéma průběhu generování IV a hashe IV

Následuje podrobný popis jednotlivých kroků.

3.2.1 Krok 1 – Prodloužení uživatelského hesla

V první kroku se při generování *IV* uživatelské heslo prodlouží tím způsobem, že se 11 krát nakopíruje za sebe.

$$Heslo_{výstup} = 11 \cdot Heslo_{vstup} \quad (19)$$

příklad pro vstupní uživatelské heslo „Ahoj“:

$$Heslo_{výstup} = 11 \cdot 'Ahoj' = 'AhojAhojAhojAhojAhojAhojAhojAhojAhojAhojAhoj' \quad (20)$$

Díky tomu se docílí prodloužení i krátkých uživatelských hesel a také dojde k navýšení bezpečnosti dalších kroků.

3.2.2 Krok 2 – Připojení defaultního řetězce

V druhém kroku se k prodlouženému heslu na konec připojí defaultní řetězec, který obsahuje méně používané a speciální znaky. Defaultní řetězec se používá i tehdy, pokud uživatel pro šifrování nezadá žádné heslo. Defaultní řetězec má následující tvar – (v jazyce Python)

```
defaultniRetezec = 'noěščščžýžýéíáňřčššéřřóů š<x>r0193ůů^i*nd-{iuilllůašůwšl'+><d--..)úůšaášžýčč( )úůš*^áiěš[]šš'
```

Obrázek 27 - Defaultní řetězec v jazyce Python

3.2.3 Krok 3 – Vmísení znaků defaultního řetězce

V třetím kroku se pomocí prodlouženého uživatelského hesla s připojeným defaultním řetězcem vmísí znaky z defaultního řetězce do hesla. Vmísení znaku defaultního řetězce probíhá vždy za každým znak prodlouženého hesla. To znamená, že se vezme znak prodlouženého hesla na pozici i , $i \in \langle 0, \text{délkaHesla} - 1 \rangle$ (Pozn.: i se počítá od nuly kvůli vlastnostem programovacího jazyka Python) a za něj se připojí znak z defaultního řetězce na pozici, která se vypočítá pomocí následujícího vzorce.

$$\text{PoziceZnaku} = (\text{ord}(\text{znakProdlouženéhoHesla}_i) \cdot i) \bmod \text{DélkaDefaultního řetězce}$$

Funkce `ord()` nám vrací pořadí znaku v ASCII tabulce (jeho decimální vyjádření binární hodnoty).

3.2.4 Krok 4 – Výpočet hashe hesla po vmísení znaků defaultního řetězce

V tomto kroku se z hesla vypočítá hash pomocí hashovací funkce SHA-256. Tím docílíme, že nebude žádný vztah mezi a uživatelským heslem, protože otisk hashovací funkce SHA-256 není možné převést zpět. Dle vyhlášky č. 378/2006 Sb. O postupech kvalifikovaných poskytovatelů certifikačních služeb ve smyslu zákona č. 227/2000 Sb. O elektronickém podpisu, můžeme SHA-256 považovat za důvěryhodnou. Vyhláška vymezuje použití standardů kryptografických algoritmů a přímo se odkazuje na použití standardu FIPS PUB 180-2, kterého je hashovací funkce SHA-256 rodiny hashovacích funkcí SHA-2 součástí. U těchto funkcí nebyly dle standardu doposud objeveny bezpečnostní slabiny. Hash hesla je uložený ve hexadecimální formě \rightarrow řetězec 64 znaků (bajtů).

3.2.5 Krok 5 – Generování IV na základě kombinování hashe hesla a hesla po vmísení znaků defaultního řetězce

V pátém kroku si ukážeme, jak se bude generovat IV na základě kombinace hashe hesla a hesla po vmísení znaků defaultního řetězce, dále jen heslo. Vygenerovaný IV nám bude sloužit pro další operace nezbytné pro chod šifrování a dále jako klíč pro šifrování prvního bloku dat. Pro následující kombinování musíme nejprve vygenerovat mapu indexů, která určí pozice znaků hashe hesla, které budou modifikovány heslem. Jako první se při generování mapy indexů určí počet dílů hesla o velikosti 16 bajtů a počet zbývajících bajtů hesla po vmísení.

$$n = l \div 16 \tag{21}$$

$$z = l \bmod 16 \quad (22)$$

- n – počet celých dílů hesla
- l – délka hesla
- div – vrací výsledek po celočíselném dělení

Dále se na základě l , počtu zbývajících bajtů a hesla vygeneruje tzv. mapa indexů. Mapa indexů je dvojrozměrné pole, které může nabývat následujících rozměrů

1. $n \cdot 16$ – pokud je l dělitelné 16 beze zbytku.
2. $(n + 1) \cdot 16$ s tím, že poslední prvek pole má velikost l – pokud je l dělitelná 16 se zbytkem.

Mapa indexů potom obsahuje celkem tolik prvků, kolik znaků (bajtů) má heslo. Nejprve se budou měnit znaky hashe hesla na pozicích dle mapy indexů modifikované heslem tolikrát, kolik je n krát 16. Modifikace probíhá pomocí následujícího vzorce, který vrací hodnotu znaku na intervalu 0..255, jedná se o případ, kdy se počítá s částmi mapy indexů vygenerovaných pro celé bloky hesla:

$$h_m = (k_{i \cdot 16 + j} + i \cdot j + h_m) \bmod 256,$$

$$m = M_{i,j}, \forall i \in \langle 0, n - 1 \rangle, \forall j \in \langle 0, 15 \rangle \quad (23)$$

- h – hodnota znak hashe hesla na intervalu 0..255
- M – mapa indexů
- m – pozice znaku hashe hesla odvozena z mapy indexů na pozici i, j
- i – index řádku mapy indexů
- j – index sloupce mapy indexů
- k – hodnota znaku hesla na intervalu 0..255
- n – počet celých bloků hesla o velikosti 16

Pokud byla délka hesla dělitelná 16 se zbytkem, musíme ještě provést další modifikaci znaků hashe hesla tolikrát, kolik bylo zbývajících znaků. Modifikace bude probíhat dle následujících vzorců:

$$h_m = (k_{i \cdot 16 + j} + i + h_m) \bmod 256,$$

$$m = M_{i,j}, i = n, \forall j \in \langle 0, z - 1 \rangle \quad (24)$$

- h – hodnota znak hashe hesla na intervalu 0..255
- M – mapa indexů
- m – pozice znaku hashe hesla odvozena z mapy indexů na pozici i, j
- i – index řádku mapy indexů
- j – index sloupce mapy indexů
- k – hodnota znaku hesla na intervalu 0..255
- n – počet celých bloků hesla o velikosti 16
- z – počet zbývajících znaků hesla po dělení 16

Jako inicializační vektor (první použitý klíč), který bude používán k šifrování, je výsledná kombinace hashe hesla a hesla.

Ještě je nutné vysvětlit, jak se generují prvky mapy indexů. Pomocí sumace hodnot bajtů klíče se určí „startovní“ hodnota s .

$$s = \sum_{i=0}^n h_i \quad (25)$$

- n – délka hesla
- h – hodnota bajtu hesla na pozici i

Při generování indexů pro celé bloky hesla se postupuje následovně. Pro každý celý blok hesla se vždy vygeneruje 16 indexů, což probíhá n -krát, podle počtu celých bloků. Vždy indexy 1, 5, 9 a 13 se vždy generují stejným vztahem, obdobně se generují indexy (2, 6, 10 a 14; 3, 7, 11 a 15; 4, 8, 12 a 16). Generování 4 indexů se řídí následovně s tím, že navýšená „startovní“ hodnota s se ukládá.

$$i_{1,5,9,13} = (s + 2) \bmod 64 \quad (26)$$

$$i_{2,6,10,14} = (s + 3) \bmod 64 \quad (27)$$

$$i_{3,7,11,15} = (s + 5) \bmod 64 \quad (28)$$

$$i_{4,8,12,16} = (s + 11) \bmod 64 \quad (29)$$

Při generování indexů pro zbývajících bajty hesla, se postupuje obdobně jako předtím, jen s tím rozdílem, že teď má každý index svůj vzorec. Navýšená „startovní“ hodnota s se opět ukládá pro další vzorce. Jelikož zbývajících blok hesla může být velikosti 1 až 15, je navrženo 15 následujících rovnic.

$$i_1 = (s + 2) \bmod 64, i_2 = (s + 7) \bmod 64, i_3 = (s + 11) \bmod 64, \quad (30,31,32)$$

$$i_4 = (s + 177) \bmod 64, i_5 = (s + 3) \bmod 64, i_6 = (s + 7) \bmod 64, \quad (33,34,35)$$

$$i_7 = (s + 111) \bmod 64, i_8 = (s + 17) \bmod 64, i_9 = (s + 19) \bmod 64, \quad (36,37,38)$$

$$i_{10} = (s + 13) \bmod 64, i_{11} = (s + 8) \bmod 64, i_{12} = (s + 29) \bmod 64, \quad (39,40,41)$$

$$i_{13} = (s + 113) \bmod 64, i_{14} = (s + 67) \bmod 64, i_{15} = (s + 31) \bmod 64 \quad (42,43,44)$$

V podstatě může říct, že generování mapy indexů se chová jako pseudonáhodný generátor posloupnosti čísel v rozmezí 0 až 63.

3.2.5.1 Zdrojový kód pro Krok 5 v jazyce Python verze 3.x

Implementace Kroku 5 do jazyka Python verze 3.x má následující zdrojový kód.

```
pocetKroku = delkaHeslaPoVmiseni//16
delkaZbytku = delkaHeslaPoVmiseni%16
mapyIndexu = GenerujMapyIndexu(pocetKroku, delkaZbytku, hesloPoVmiseni)
indexVHeslePoVmiseni = 0
IVList = list(hashHeslaPoVmiseni)
for i in range(0, pocetKroku):
    for j in range(0, 16):
        pozice = mapyIndexu[i][j]
        IVList[pozice] = chr((ord(hesloPoVmiseni[indexVHeslePoVmiseni]) + j*i + ord(hashHeslaPoVmiseni[pozice])) % 256)
        indexVHeslePoVmiseni += 1
for i in range(0, delkaZbytku):
    pozice = mapyIndexu[pocetKroku][i]
    IVList[pozice] = chr((ord(hesloPoVmiseni[indexVHeslePoVmiseni]) + i + ord(hashHeslaPoVmiseni[pozice])) % 256)
    indexVHeslePoVmiseni += 1
IV = ''
for i in range(0, len(IVList)):
    IV += IVList[i]
```

Obrázek 28 - Zdrojový kód pro Krok 5

Pro úplnost je potřeba ukázat zdrojový kód funkce pro vygenerování map indexů pro vmísení hesla do hashe hesla.

```
def GenerujMapyIndexu(pocet, zbytek, klic):
    mapy = []
    start = 0
    for i in range(0, len(klic)):
        start += ord(klic[i])
    if pocet > 0:
        for i in range(0, pocet):
            mapyCast = []
            for j in range(0, 4):
                start += 2
                mapyCast.append(start%64)
                start += 3
                mapyCast.append(start%64)
                start += 5
                mapyCast.append(start%64)
                start += 11
                mapyCast.append(start%64)
            mapy.append(mapyCast)
    mapyCast = []
    for i in range(0, zbytek):
        if i == 0:
            start += 2
        if i == 1:
            start += 7
        if i == 2:
            start += 11
        if i == 3:
            start += 177
        if i == 4:
            start += 3
        if i == 5:
            start += 7
        if i == 6:
            start += 111
        if i == 7:
            start += 17
        if i == 8:
            start += 19
        if i == 9:
            start += 13
        if i == 10:
            start += 8
        if i == 11:
            start += 29
        if i == 12:
            start += 113
        if i == 13:
            start += 67
        if i == 14:
            start += 31
        mapyCast.append(start%64)
    mapy.append(mapyCast)
    return(mapy)
```

*Obrázek 29 - Zdrojový kód
funkce pro generování mapy
indexů*

3.2.6 Krok 6 – Výpočet hashe

Na závěr se opět pomocí hashovací funkce SHA-256 vypočítá hash , který je používán k autorizaci uživatele při dešifrování.

3.3 Návrh pomocné funkce pro zamíchání substituční tabulky

Pro substituci při šifrování a dešifrování je využívána substituční tabulka, která do procesu vstupuje jako konstanta. Substituční tabulka je pole o velikosti 256 prvků a obsahuje hodnoty od 0 do 255. Je reprezentací hodnot všech znaků, které mohou mít jeden bajt. Substituční tabulka má na začátku následující tvar, viz. Příloha I – Substituční tabulka. Tudíž je vygenerovaná ještě před samotných šifrováním nebo dešifrováním. O účely zajištění jedinečnosti substituční tabulky pro šifrování a dešifrování byla navržena funkce pro

zamíchání substituční tabulky, která vrací substituční tabulku se zamíchaných pořadím hodnot. Míchání probíhá na základě klíče (popřípadě na začátku na základě IV). Zamíchání substituční tabulky se provádí na začátku před šifrováním prvního bloku dat a potom vždy mezi šifrováním dalšího bloku dat. Substituční tabulka, slouží pro operaci substituce při šifrování bloku dat. Princip činnosti substituce bude rozebrán v další kapitole. Nyní k principu fungování funkce pro zamíchání substituční tabulky. Funkce vezme substituční tabulku a poté každý prvek na pozicích 0..255 vymění s jiným prvkem na pozici, která je určena pomocí klíče (IV – před šifrováním prvního bloku dat). Prvek substituční tabulky s_i se vymění s prvkem substituční tabulky s_j . Hodnota pozice j je vypočítaná ze vztahu

$$j = k_i \bmod 64 \quad (45)$$

- k – hodnota znaku klíče (IV)
- i – index prvního znaku substituční tabulky
- j – index druhého znaku substituční tabulky

3.3.1 Zdrojový kód funkce pro zamíchání substituční tabulky

Funkce pro zamíchání substituční tabulky je pojmenovaná *ZamichaniSubstitucniTabulky()*. Funkce zajišťuje zamíchání substituční tabulky v závislosti na klíči. Funkce má dva vstupní parametry, první je *substitucniTabulkaIn* – vstupní substituční tabulka a druhý je *klic* – vstupní klíč pro zamíchání substituční tabulky. Funkce vrací proměnnou *substitucniTabulkaOut*, která obsahuje zamíchanou substituční tabulku pomocí klíče. Implementace navržené funkce v jazyce Python verze 3.x vypadá následovně.

```
def ZamichaniSubstitucniTabulky(substitucniTabulkaIn, klic):
    substitucniTabulkaOut = substitucniTabulkaIn[:]
    for i in range(0, 256):
        pom = substitucniTabulkaOut[i]
        substitucniTabulkaOut[i] = substitucniTabulkaOut[ord(klic[i % 64])]
        substitucniTabulkaOut[ord(klic[i % 64])] = pom
    return substitucniTabulkaOut
```

Obrázek 30 - Zdrojový kód funkce pro zamíchání substituční tabulky

3.4 Návrh operací pro šifrování

V této části si popíšeme navržené operace pro šifrování. Jedná se o jeden z nejdůležitějších prvků celého návrhu. V kapitole jsou popsány operace pro substituci, pro přičtení klíče k datům a pro transpozici. Všechny jsou navrženy tak, aby jejich činnost byla modifikovaná hodnotou klíče. Všechny operace jsou založeny na principu výše popsané

operace sečtení modulo. Operace sčítání modulo je využita nejen k sčítání a modulování hodnot bajtů vstupních dat a klíče, ale také k určování indexů pro operace na vstupních datech. Operace jsou navrženy tak, aby bylo možné ze znalosti výstupních (zašifrovaných) dat a klíče odvodit pomocí operací pro dešifrování zpět vstupní data. Operace mají tzv. svojí inverzní operaci. Inverzní operace k sečtení modulo byla popsána v kapitole 1.2.1.2. *Operace sčítání modulo.*

3.4.1 Substituce

Jak již bylo uvedeno na začátku diplomové práce, substituce slouží k nahrazení znaku otevřeného textu (bajtem nezašifrovaných dat) znakem zašifrovaného textu (bajtem zašifrovaných dat) na základě stanovených pravidel. My pro substituci budeme využívat substituční tabulku a klíč. Bajt vstupního bloku dat bude na základě klíče substituován bajtem ze substituční tabulky na bajt zašifrovaných dat. Při substituci se vezme bajt bloku dat k šifrování na pozici i a odpovídající bajt klíče na pozici i . Z hodnot těchto bajtů se určí pozice s bajtu v substituční tabulce. Následně je bajt nezašifrovaného bloku dat na pozici i nahrazen hodnotou bajtu substituční tabulky na pozici s . Výpočet pozice s se provádí následující vztah

$$s = (b_i + k_i) \bmod 256, \forall i \in \langle 0, 255 \rangle \quad (46)$$

- s – pozice bajtu substituční tabulky
- b – hodnota bajtu bloku nezašifrovaných dat na pozici i
- k – hodnota bajtu klíče na pozici i

Při substituci se prochází vstupní blok dat zleva doprava \rightarrow od pozice 0 do pozice 63.

3.4.1.1 Zdrojový kód funkce pro substituci

Implementace navržené funkce pro substituci do jazyka Python verze 3.x má tento zdrojový kód.

```
def Substituce(blokKSifrovani, klic, substitucniTabulkaIn):
    zasifrovanyBlok = ''
    for i in range(0, delkaBloku):
        index = (ord(blokKSifrovani[i]) + ord(klic[i])) % 256
        zasifrovanyBlok += chr(substitucniTabulkaIn[index])
    return zasifrovanyBlok
```

Obrázek 31 - Zdrojový kód funkce pro substituci

3.4.2 Operace Add – přičtení klíče k datům

Druhá operace, která se aplikuje na každý blok vstupních dat je operace přičtení klíče. Přičtení se opět řídí klíčem a provádí se pomocí operace sčítání modulo. Přičítání probíhá pro každý bajt vstupních dat. Blok vstupních dat o velikosti 64 bajtů (označme si jako n) se prochází bajt po bajtu, index označující pozici bajtu vstupních dat si označme c a index odpovídajícího bajtu klíče označme jako j . Při procházení se $c=j$. Dále mohou nastat dva případy.

1. Hodnota bajtu klíče na j -té je sudá. $\rightarrow k_j \bmod 2 = 0$
2. Hodnota bajtu klíče na j -té pozici je lichá. $\rightarrow k_j \bmod 2 = 1$

V prvním případě se hodnota bajtu klíče na pozici j přičítá ke všem bajtům vstupních dat na pozicích doprava od pozice c včetně. Operaci lze vyjádřit následovně.

$$b_i = (b_i + k_j) \bmod 256, \forall i \in \langle c, n-1 \rangle \quad (47)$$

V druhém případě se hodnota bajtu klíče na pozici j přičítá ke všem bajtům vstupních dat na pozicích doleva od pozice c včetně. Operaci můžeme vyjádřit obdobně.

$$b_i = (b_i + k_j) \bmod 256, \forall i \in \langle 0, c \rangle \quad (48)$$

3.4.2.1 Zdrojový kód funkce pro Add – přičtení klíče k datům

Implementace navržené funkce pro přičtení klíče k datům do jazyka Python verze 3.x má tento zdrojový kód.

```
def Add(blokKSifrovani, klic):
    zasifrovanyBlokList = list(blokKSifrovani)
    for i in range(0, delkaBloku):
        aktZnakKlice = klic[i]
        aktZnakBloku = zasifrovanyBlokList[i]
        if ord(aktZnakKlice) % 2 == 0:
            for j in range(i, delkaBloku):
                zasifrovanyBlokList[j] = chr((ord(zasifrovanyBlokList[j]) + ord(aktZnakKlice)) % 256)
        else:
            for j in range(i, -1, -1):
                zasifrovanyBlokList[j] = chr((ord(zasifrovanyBlokList[j]) + ord(aktZnakKlice)) % 256)
    zasifrovanyBlokStr = ''
    for i in range(0, delkaBloku):
        zasifrovanyBlokStr += zasifrovanyBlokList[i]
    return zasifrovanyBlokStr
```

Obrázek 32 - Zdrojový kód funkce Add - přičtení klíče

3.4.3 Mixování - transpozice

Operace slouží k přeházení pořadí znaků vstupního bloku dat. Jedná se o transpozici, která je řízena klíčem. Vstupní blok dat se postupně prochází bajt za bajtem a pro každý bajt na pozici i je klíčem na odpovídající pozici i vypočtena pozice j , na které je bajt dat,

ten se prohodí s bajtem vstupních bloku dat na pozici i . Pozice znaku k prohození j je spočtena následovně.

$$j = k_i \bmod 64 \quad (49)$$

- k_i – hodnota bajtu klíče na pozici i

3.4.3.1 Zdrojový kód funkce pro mixování

Implementace navržené funkce pro mixování do jazyka Python verze 3.x má tento zdrojový kód.

```
def Mixovani(blokKSifrovani, klic):
    zasifrovanyBlokList = list(blokKSifrovani)
    for i in range(0, delkaBloku):
        mixIndex = ord(klic[i]) % delkaBloku
        zasifrovanyBlokList[i], zasifrovanyBlokList[mixIndex] = zasifrovanyBlokList[mixIndex], zasifrovanyBlokList[i]
    zasifrovanyBlokStr = ''
    for i in range(0, delkaBloku):
        zasifrovanyBlokStr += zasifrovanyBlokList[i]
    return zasifrovanyBlokStr
```

Obrázek 33 - Zdrojový kód funkce pro mixování

3.5 Návrh operací pro dešifrování

Pro každou operaci, která zajišťuje šifrování, je navržená operace zajišťující dešifrování. Operace pro dešifrování můžeme nazývat inverzními operacemi k operacím pro šifrování. Bylo nutné navrhnout inverzní operaci k substituci, k přičítání klíče a k mixování. Nyní si ukažme, jak fungují.

3.5.1 „DeSubstitute“ – inverzní operace k substituci

Průběh operace „DeSubstitute“ je ze všech tří nejméně podobný s operací pro šifrování. Při jediné se blok vstupních zašifrovaných dat neprochází zprava doleva, ale opět jako při substituci zleva doprava → od indexu 0 po index 63. Do operace vstupuje substituční tabulka v pořadí, jako měla před substitucí při šifrování. Nejprve se v substituční tabulce vyhledá pozice s znaku, který má odpovídající hodnotu bajtu vstupních zašifrovaných dat na pozici i . Následně se vypočítá hodnota bajtu dešifrovaných dat d na pozici i ze vztahu.

$$d_i = (s - k_i) \bmod 256 \quad (50)$$

- d_i – hodnota bajtu výstupních (dešifrovaných) dat na pozici i
- s – index hodnoty bajtu vstupních dat v substituční tabulce
- k_i - hodnota bajtu klíče na pozici i

Můžeme si všimnout, že při inverzní operaci k substituci se prohledává pole o velikost 256 znaků, což je bohužel brzdícím faktorem celého průběhu dešifrování, protože vyhledávání má složitost $O(n)$ na rozdíl od přístupu do pole, které díky dobré implementaci konstantní.

3.5.1.1 Zdrojový kód funkce pro inverzní operaci k substituci

Implementace navržené funkce pro inverzní operaci k substituci do jazyka Python verze 3.x má tento zdrojový kód.

```
def DeSubstitute(blokKDesifrovani, klic, substitucniTabulkaIn):
    desifrovanyBlok = ''
    for i in range(0, delkaBloku):
        index = substitucniTabulkaIn.index(ord(blokKDesifrovani[i]))
        desifrovanyBlok += chr((index - ord(klic[i])) % 256)
    return desifrovanyBlok
```

Obrázek 34 - Zdrojový kód funkce pro inverzní operaci k substituci

3.5.2 „DeAdd“ – inverzní operace k Add – odečtení klíče od dat

Tato inverzní operace k operaci Add je obdobná jako samotná operace Add, akorát vstupní zašifrovaný blok dat se prochází zprava doleva → od indexu 63 po index 0. A místo operace sčítání modulo se používá odčítání modulo. Určující pozici bajtu vstupních dat k dešifrování si opět označme jako c . Hodnotu bajtu vstupních dat si označme jako b . Operace je opět rozdělena na dvě možnosti chování.

1. Hodnota bajtu klíče na j -té je sudá. → $k_j \bmod 2 = 0$
2. Hodnota bajtu klíče na j -té pozici je lichá. → $k_j \bmod 2 = 1$

V prvním případě se hodnota bajtu klíče na pozici j odečítá od všech bajtů vstupních dat na pozicích doprava od pozice c včetně. Operaci lze vyjádřit následovně.

$$b_i = (b_i - k_j) \bmod 256, \forall i \in \langle c, n - 1 \rangle \quad (51)$$

V druhé případě se hodnota bajtu klíče na pozici j odčítá ke všem bajtům vstupních dat na pozicích doleva od pozice c včetně. Operaci můžeme vyjádřit obdobně.

$$b_i = (b_i - k_j) \bmod 256, \forall i \in \langle 0, c \rangle \quad (52)$$

3.5.2.1 Zdrojový kód funkce pro inverzní operaci k Add – odečtení klíče

Implementace navržené funkce pro inverzní operaci k Add – odečtení klíče do jazyka Python verze 3.x má tento zdrojový kód.

```
def DeAdd(blokKDesifrovani,klic):
    desifrovanyBlokList = list(blokKDesifrovani)
    for i in range(delkaBloku - 1,-1,-1):
        aktZnakKlice = klic[i]
        aktZnakBloku = desifrovanyBlokList[i]
        if ord(aktZnakKlice) % 2 == 0:
            for j in range(i,delkaBloku):
                desifrovanyBlokList[j] = chr((ord(desifrovanyBlokList[j]) - ord(aktZnakKlice)) % 256)
        else:
            for j in range(i,-1,-1):
                desifrovanyBlokList[j] = chr((ord(desifrovanyBlokList[j]) - ord(aktZnakKlice)) % 256)
    desifrovanyBlokStr = ''
    for i in range(0,delkaBloku):
        desifrovanyBlokStr += desifrovanyBlokList[i]
    return desifrovanyBlokStr
```

Obrázek 35 - Zdrojový kód funkce pro inverzní operaci k Add - odečtení klíče

3.5.3 „DeMixování“ – inverzní operace k mixování

Třetí z operací má obdobnou funkci jako původní operace pro šifrování mixováním. Probíhá naprosto stejně s tím rozdílem, že se vstupní blok zašifrovaných dat prochází zprava doleva → od indexu 63 po index 0. Jinak se opět z hodnoty bajtu klíče na pozici i vypočítá pozice bajtu j . Následně dochází v prohození bajtů vstupních dat na pozicích i a j . Výpočet pozice j má následující tvar.

$$j = k_i \bmod 64 \quad (53)$$

- k_i – hodnota bajtu klíče na pozici i

3.5.3.1 Zdrojový kód funkce pro inverzní operaci k mixování

Implementace navržené funkce pro inverzní operaci k mixování do jazyka Python verze 3.x má tento zdrojový kód.

```
def DeMixovani(blokKDesifrovani,klic):
    desifrovanyBlokList = list(blokKDesifrovani)
    for i in range(delkaBloku - 1,-1,-1):
        mixIndex = ord(klic[i]) % delkaBloku
        desifrovanyBlokList[i],desifrovanyBlokList[mixIndex] = desifrovanyBlokList[mixIndex],desifrovanyBlokList[i]
    desifrovanyBlokStr = ''
    for i in range(0,delkaBloku):
        desifrovanyBlokStr += desifrovanyBlokList[i]
    return desifrovanyBlokStr
```

Obrázek 36 - Zdrojový kód funkce pro inverzní operaci k mixování

3.6 Operace pro určení posloupnosti operací pro šifrování a dešifrování

Již na začátku kapitoly 3, které se věnuje popisu prvků a vlastnostem navržené šifry, byla jedna z vlastností ta, že se pořadí operací – Substitute, Add (přičtení klíče) a Mixování bude řídit klíčem. Navržená šifra nemá pevnou strukturu šifrování, jak je tomu například u šifry AES (SubByte Transformation, ShiftRow Transformation, MixColumn Transformation a Add Round Key), ale posloupnost operací se bude generovat v závislosti na klíči. Pravidlem je ovšem to, aby během jedné rundy byly použity všechny 3 operace,

což nám dává pro jednu rundu 6 možných posloupností operací. Pokud bychom si základní posloupnost 3 operací označili písmeny ,SAM‘.

- S – Substituce
- A – Add – přičtení klíče
- M – Mixování

Dostaneme následující kombinace písmen udávající posloupnost operací pro jednu rundu.

- SAM, SMA, ASM, AMS, MAS a MSA

Pokud bude šifrování probíhat v 8 rundách, tak dostáváme $6^8 = 1679616$ možných kombinací posloupností operací, jak šifrovat. Což znesnadňuje kryptoanalýzu šifry pomocí metod navržených pro šifry s pevnou strukturou. Počet posloupností operací p pro n rund se vypočítá následujícím vzorcem.

$$p = 6^n \quad (54)$$

Pro dešifrování se používá převrácená posloupnost operací. To znamená, že pokud se šifrovalo v 8 rundách posloupností operací zapsané jako SAM-SAM-SAM-SAM-SAM-SAM-SAM-SAM, tak pro dešifrování je nutno použít reverzní posloupnost MAS-MAS-MAS-MAS-MAS-MAS-MAS-MAS. A nyní k popisu generování posloupnosti operací. Operace probíhá podobně, jako při generování mapy indexů v kapitole 3.2.5. Do operace vstupují klíč a počet rund. Hodnotu bajtu klíče si označme jako k a počet rund jako n . Celý proces je nastartovaný hodnotou bajtu klíče na pozici 11, která se uloží jako startovní hodnota $s \rightarrow s = k_{11}$. Jedním krokem generování posloupnosti se vygeneruje posloupnost pro jednu rundu, takže celý proces generování probíhá v n krocích. Posloupnost pro i -tou rundu probíhá v následujících krocích.

1. Jako klíč pro generování posloupnosti pro rundu je 8 bajtů z klíče pro šifrování. Podklíč je složený z 8 po sobě jdoucích bajtů na pozicích x až y . Pro výpočet mezi x a y se používají následující vztahy.

$$x = (i \bmod 8) \cdot 8 \quad (55)$$

$$y = ((i \bmod 8) \cdot 8) + 8 \quad (56)$$

2. Navýšení startovací hodnoty s pomocí hodnoty klíče k na pozici j hodnoty k na pozici l . Hodnoty j a l dostaneme z následujících vztahů.

$$j = (11 \cdot s) \bmod 64 \quad (57)$$

$$l = i \bmod 64 \quad (58)$$

Hodnota s bude vypočtena následovně.

$$s = s + k_j + k_l + i \quad (59)$$

3. Ještě dochází k úpravě hodnoty s a to sumací hodnot bajtů klíče na pozicích x až y .

$$s = s + \sum_{a=x}^y k_a \quad (60)$$

4. Následně se vypočítá hodnota z , podle které se určí jedna ze šesti posloupností pro rundu i . Rovnice pro výpočet z vypadá následovně.

$$z = s \bmod 6 \quad (61)$$

5. Nakonec se přiřadí posloupnost operací pro rundu i z hodnoty z následující tabulkou

Tabulka 2 - Přiřazení posloupností pro i -tou rundu

z	posloupnost pro rundu i
0	SAM
1	ASM
2	SMA
3	AMS
4	MSA
5	MAS

3.6.1 Zdrojový kód funkce pro vygenerování posloupnosti operací

Implementace navržené funkce pro vygenerování posloupnosti operací použitých při šifrování/dešifrování do jazyka Python verze 3.x má tento zdrojový kód.

```
def VygenerujSledOperaci(klic, pocetRund):
    sledOperaci = ''
    soucet = ord(klic[11])
    for i in range(0, pocetRund):
        klicPart = '' + klic[(i%8)*8:((i%8)*8) + 8]
        soucet += ord(klic[(11*soucet) % 64]) + i + ord(klic[i%64])
        for j in range(0, 8):
            soucet += ord(klicPart[j])
        zbytek = soucet % 6
        if zbytek == 0:
            sledOperaci += 'sam' #Sled operací - Substituce, Add, Mixování
        if zbytek == 1:
            sledOperaci += 'asm' #Sled operací - Add, Substituce, Mixování
        if zbytek == 2:
            sledOperaci += 'mas' #Sled operací - Mixování, Add, Substituce
        if zbytek == 3:
            sledOperaci += 'ams' #Sled operací - Add, Mixování, Substituce
        if zbytek == 4:
            sledOperaci += 'msa' #Sled operací - Mixování, Substituce, Add
        if zbytek == 5:
            sledOperaci += 'sma' #Sled operací - Substituce, Mixování, Add
    return sledOperaci
```

Obrázek 37 - Zdrojový kód funkce pro vygenerování posloupnosti operací

3.7 Režim činnosti – operace pro posun klíče

Jak bylo napsané na začátku kapitoly, tak navržená šifra nemá žádný ze známých režimů činnosti, které využívají jiné šifry (AES, DES, apod.), ale režim činnosti a posun klíče s tím spojený se řídí klíčem, který byl předtím použitý pro šifrování/dešifrování bloku dat. To znamená, že k posunutí klíče dochází vždycky po zašifrování jednoho bloku dat, aby další blok byl šifrován jiným klíčem odvozeným od IV . Nový klíč se určuje sice z hodnot bajtů vstupních dat, bajtů výstupních dat a bajtů klíče, ale co a kdy se používá k výpočtu se řídí hodnotou klíče. Nový klíč se skládá z 64 nově vygenerovaných bajtů. Kdy každý i -tý bajt nového klíče je vypočítán následovně. Nejprve stanovíme hodnoty.

- o – hodnota bajtu vstupních (nezašifrovaných) dat na pozici i
- s – hodnota bajtu výstupních (zašifrovaných) dat na pozici i
- k – hodnota bajtu posledního použitého klíče na pozici i
- n – startovní hodnota bajtu nového klíče

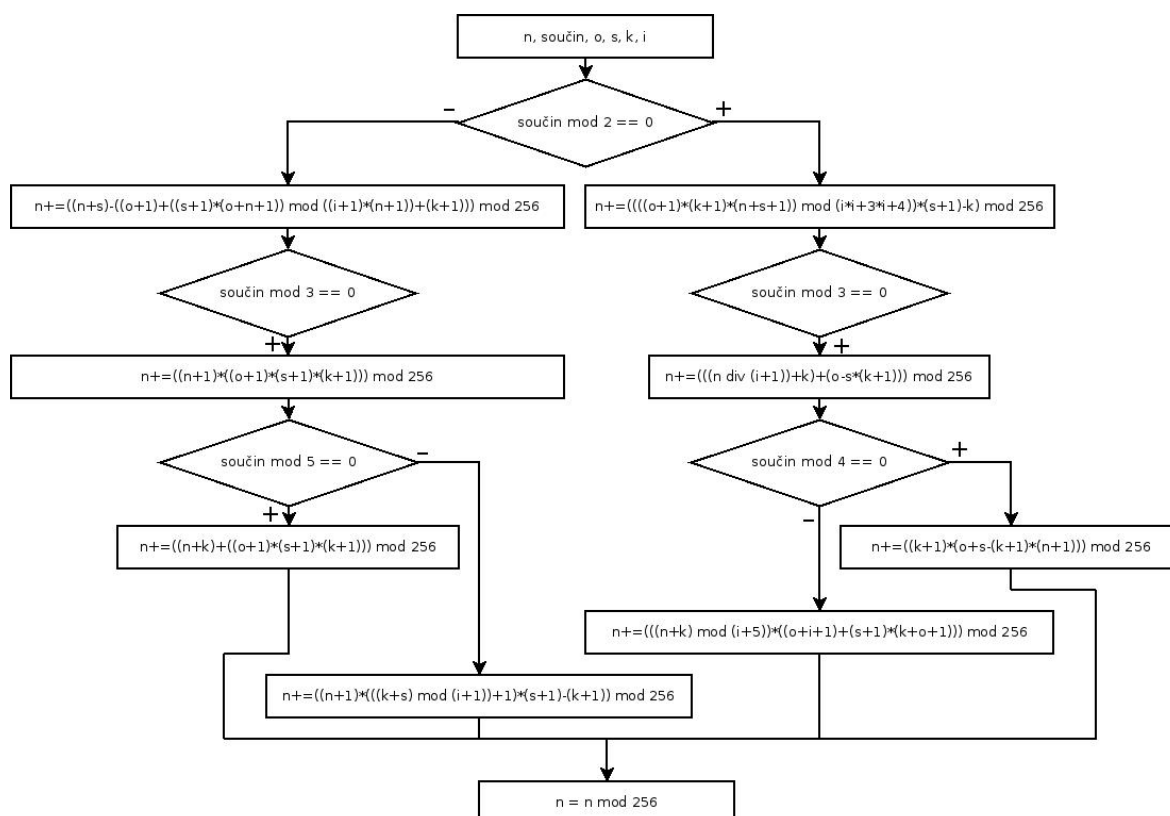
Startovní hodnota n se vypočítá následovně

$$n = k_{(i+11) \bmod 64} \quad (62)$$

Dále spočítáme součin hodnot o , s a k , v závislosti na kterém se bude měnit hodnota bajtu nového klíče n . Výpočet součinu vypadá následovně

$$\text{součin} = (o + 1) \cdot (s + 1) \cdot (k + 1) \quad (63)$$

Hodnoty o , s a k leží na intervalu 0..255. Proto se zvětšují o 1, aby nedocházelo k vynulování součinu. Změny hodnoty n dle součinu ukazuje následující vývojový diagram.

Obrázek 38 – Vývojový diagram pro určení hodnoty i -tého bajtu nového klíče

Poznámka k vývojovému diagramu – operace div vrací hodnotu po celočíselném dělení.

3.7.1 Zdrojový kód funkce pro posunutí klíče

Implementace navržené funkce pro posunutí klíče do jazyka Python verze 3.x má tento zdrojový kód.

```

def PosunKlic(aktBlokOtevreny, aktBlokSifrovany, aktKlic, delkaBloku = 64):
    novyKlic = ''
    for i in range(0, delkaBloku):
        o = aktZnakBlokOtevrenyInt = ord(aktBlokOtevreny[i])
        s = aktZnakBlokSifrovanyInt = ord(aktBlokSifrovany[i])
        k = aktZnakKlicInt = ord(aktKlic[i])
        n = novyZnakKlicInt = ord(aktKlic[(11 + i)%64])
        soucin = (o + 1) * (s + 1) * (k + 1)
        if soucin % 2 == 0:
            n += (((o + 1) * (k + 1) * (n + s + 1)) % ((i * 3 + 4) * (s + 1) - k)) % 256
            if soucin % 3 == 0:
                n += (((n // (i + 1)) + k) + (o - s * (k + 1))) % 256
                if soucin % 4 == 0:
                    n += ((k + 1) * (o + s - (k + 1) * (n + 1))) % 256
                else:
                    n += (((n + k) % (i + 5)) * ((o + i + 1) + (s + 1) * (k + o + 1))) % 256
            else:
                n += ((n + s) - ((o + 1) + ((s + 1) * (o + n + 1)) % ((i + 1) * (n + 1) + (k + 1)))) % 256
                if soucin % 3 == 0:
                    n += ((n + 1) * (o + 1) * (s + 1) * (k + 1)) % 256
                    if soucin % 5 == 0:
                        n += ((n + k) + ((o + 1) * (s + 1) * (k + 1))) % 256
                    else:
                        n += ((n + 1) * (((k + s) % (i + 1)) + 1) * (s + 1) - (k + 1)) % 256
                novyKlic += chr(n % 256)
    return novyKlic

```

Obrázek 39 - Zdrojový kód funkce pro posunutí klíče

3.8 Pomocná funkce pro zakódování bajtu udávající počet doplněných znaků (bajtů) necelého bloku dat

Při šifrování souboru blokovou šifrou mohou nastat případy, kdy je potřeba zašifrovat necelý blok dat, protože všechny soubory nemají velikost, která je násobkem velikosti bloku dat. Z tohoto důvodu bylo navrženo doplňování necelého bloku dat náhodnými znaky. Aby se dalo při dešifrování kolik znaků (bajtů) bylo doplněno se nakonec souboru ukládá bajt udávající počet doplněných znaků. Aby nebylo pro útočníka jednoduché určit počet doplněných znaků při šifrování a tudíž délku posledního bloku, byla navržena funkce, která tento bajt před uložením zakóduje pomocí klíče použitého pro šifrování posledního bloku dat (pokud je soubor menší jak 64 bajtů, tak se vezme IV). K hodnotě bajtu udávající počet doplněných znaků se postupně přičtou hodnoty všech bajtů klíče. Výsledek se upraví modulem 256. Úprava modulem 256 je nutná z toho důvodu, aby byla hodnota v rozmezí 0..255. Vzorec vracející hodnotu zakódovaného bajtu udávající počet doplněných znaků vypadá následovně.

$$c = \left(e + \sum_{i=0}^{63} k_i \right) \bmod 256 \quad (64)$$

- c – zakódovaná hodnota bajtu udávající počet doplněných znaků
- e – nezakódovaná hodnota bajtu udávající počet doplněných znaků
- k_i – hodnota bajtu klíče na pozici i

3.8.1 Zdrojový kód funkce pro zakódování bajtu udávající počet doplněných znaků

Implementace navržené funkce pro zakódování bajtu udávající počet doplněných znaků (bajtů) necelého bloku dat do jazyka Python verze 3.x má tento zdrojový kód.

```
def ZakodujDelkuDoplnku(delkaDoplnku,klic):  
    suma = 11  
    for i in range(0,delkaBloku):  
        suma += ord(klic[i])  
    suma += delkaDoplnku  
    return suma % 256
```

Obrázek 40 - zdrojový kód funkce pro
zakódování bajtu udávající počet dopl-
něných znaků

3.9 Funkce pro dekódování bajtu udávajícího počet doplněných znaků (bajtů) necelého bloku dat

Pokud bylo nutné při šifrování doplnit blok o určitý počet znaků, tak se musí při dešifrování poslední blok zase o určitý počet znaků zkrátit. Pro účely uložení hodnoty počtu doplněných znaků jsme si v předchozí kapitole popsali funkci, které zakóduje hodnotu bajtu udávající počet doplněných znaků. Pro dekódování byla navržena inverzní funkce k funkci pro zakódování. Funkce načte zakódovanou hodnotu udávající počet doplněných znaků a opět vezme klíč použitý k dešifrování posledního bloku (pokud je soubor menší jak 64 bajtů, tak se vezme IV) a hodnoty všech bajtů klíče se odečtou od hodnoty zakódovaného bajtu udávající počet doplněných znaků. Nakonec se hodnota upraví modulem 256. Vzorec pro dekódování hodnoty bajtu udávající počet doplněných znaků má následující tvar.

$$e = \left(c - \sum_{i=0}^{63} k_i \right) \bmod 256 \quad (65)$$

- c – zakódovaná hodnota bajtu udávající počet doplněných znaků
- e – dekódovaná hodnota bajtu udávající počet doplněných znaků
- k_i – hodnota bajtu klíče na pozici i

3.9.1 Zdrojový kód funkce pro dekódování bajtu udávající počet doplněných znaků

Implementace navržené funkce pro dekódování bajtu udávající počet doplněných znaků (bajtů) necelého bloku dat do jazyka Python verze 3.x má tento zdrojový kód.


```
def DekodujDelkuDoplнку(delkaDoplнкуKod, klic):
    suma = 11
    for i in range(0, delkaBloku):
        suma += ord(klic[i])
    return ((delkaDoplнкуKod - suma) % 256)
```

Obrázek 41 - zdrojový kód funkce pro dekódování bajtu udávající počet doplněných znaků

3.10 Funkce pro zakódování počtu rund

Protože byla šifra navržena tak, že umožňuje různý počet rund, tak bylo nutné tento počet nějakým způsobem ukládat. Z důvodu toho, aby útočník hned nepoznal kolik bylo použito run k zašifrování, tak se hodnota před uložení zakóduje pomocí *IV* a uloží na začátek zašifrovaného souboru. Funkce funguje naprosto stejně jako funkce pro zakódování bajtu udávajícího počet doplněných znaků necelého bloku dat. Akorát s tou výjimkou, že se k zakódování využívá vždy *IV*. Vzorec vracející zakódovanou hodnotu bajtu udávající počet rund má následující tvar.

$$r_c = \left(r_e + \sum_{i=0}^{63} k_i \right) \bmod 256 \quad (66)$$

- r_c – zakódovaná hodnota bajtu udávající počet rund
- r_e – nezakódovaná hodnota bajtu udávající počet rund
- k_i – hodnota bajtu *IV* na pozici i

3.10.1 Zdrojový kód funkce pro zakódování počtu rund

Implementace navržené funkce pro zakódování počtu rund do jazyka Python verze 3.x má tento zdrojový kód.

```
def ZakodujPocetRund(pocetRund, klic):
    suma = 11
    for i in range(0, delkaBloku):
        suma += ord(klic[i])
    suma += pocetRund
    return suma % 256
```

Obrázek 42 - zdrojový kód funkce pro zakódování počtu rund

I když je zdrojový kód prakticky stejný jako zdrojový kód funkce pro zakódování bajtu udávajícího počet doplněných znaků (bajtů), tak byla funkce implementovaná znovu, kdyby došlo ke změně způsobu zakódování jedné z nich.

3.11 Funkce pro dekodování počtu rund

Jedná se opět o stejnou funkci, která se využívá k dekodování hodnoty bajtu udávající počet doplněných znaků s tím rozdílem, že se vždy k dekodování používá IV . Počet rund se získá odečtení sumy hodnot bajtů IV od hodnoty bajtu zakódovaného počtu rund. Hodnotu získáme z následujícího vzorce.

$$r_e = \left(r_c - \sum_{i=0}^{63} k_i \right) \bmod 256 \quad (67)$$

- r_c – zakódovaná hodnota bajtu udávající počet rund
- r_e – dekodovaná hodnota bajtu udávající počet rund
- k_i – hodnota bajtu IV na pozici i

3.11.1 Zdrojový kód funkce pro dekodování počtu rund

Implementace navržené funkce pro dekodování počtu rund do jazyka Python verze 3.x má tento zdrojový kód.

```
def DekodujPocetRund(pocetRundKod, klic):
    suma = 11
    for i in range(0, delkaBloku):
        suma += ord(klic[i])
    return ((pocetRundKod - suma) % 256)
```

*Obrázek 43 - zdrojový kód funkce
pro dekodování počtu rund*

3.12 Velikost zašifrovaného souboru s příponou .encrypt

Velikost výstupního zašifrovaného souboru s příponou .encrypt závisí na velikosti vstupního souboru. Mohou nastat tyto dvě varianty:

1. Vstupní soubor pro zašifrování má velikost, která je násobkem 64 bajtů $\rightarrow n \cdot 64$ bajtů.
2. Velikost vstupního souboru pro zašifrování není dělitelná 64 beze zbytku \rightarrow je nutné doplnit poslední blok na velikost 64 bajtů.

Pro první případ bude mít rovnice pro výpočet výsledné velikosti souboru následující tvar:

$$v = v_v + v_h + p_r \quad (68)$$

- v – Velikost souboru po zašifrování v bajtech
- v_v – Velikost vstupního souboru pro šifrování v bajtech
- v_h – Velikost hashe uživatelského hesla v bajtech (vždy 64 bajtů)
- p_r – Bajt udávající zakódovaný počet rund použitých k zašifrování

Například – Vstupní soubor o velikost 128 bajtů, bude mít po zašifrování velikost:

$$v = 128 + 64 + 1 = 193 B \quad (69)$$

V druhém případě, kdy bude potřeba doplnit necelý blok, se nám rovnice změní na tvar:

$$v = (v_v \text{ div } 64) \cdot 64 + (v_v \text{ mod } 64) + (64 - (v_v \text{ mod } 64)) + v_h + p_r + p_{dz} \quad (70)$$

- v – Velikost souboru po zašifrování v bajtech
- v_v – Velikost vstupního souboru pro šifrování v bajtech
- v_h – Velikost hashe uživatelského hesla v bajtech (vždy 64 bajtů)
- p_r – Bajt udávající zakódovaný počet rund použitých k zašifrování
- p_{dz} – Bajt udávající zakódovaný počet doplněných znaků (bajtů)
- div – Operace pro celočíselné dělení
- mod – Operace modulo

Například – Vstupní soubor o velikost 127 bajtů, bude mít po zašifrování velikost:

$$\begin{aligned} v &= (127 \text{ div } 64) \cdot 64 + (127 \text{ mod } 64) + (64 - (127 \text{ mod } 64)) + 64 + 1 + 1 = \\ &= 1 \cdot 64 + 63 + (64 - 63) + 64 + 1 + 1 = 194 B \end{aligned} \quad (71)$$

Z toho vyplývá, že výstupní soubor po šifrování s příponou .encrypt bude mít velikost o 65 bajtů větší v případě bez doplňování znaků(bajtů) a o 66 bajtů větší v případě, kdy byla potřeba doplnit posledního necelý bloky o znaky(bajty). Doplňovaný soubor bude mít o jeden bajt víc, než nedoplňovaný soubor se stejným počtem bloků.

Tohoto se využívá na začátku dešifrování, kdy se zjišťuje, jestli byl soubor při šifrování doplněný o znaky(bajty) nebo ne. Stačí vzít velikost zašifrovaného souboru, zjistit zbytek po dělení 64 a pokud je zbytek roven 1, tak soubor nebyl při šifrování doplněný, pokud je zbytek roven 2, tak byl doplněný.

4 ANALÝZA RYCHLOSTI A VÝPOČETNÍ SLOŽITOSTI NAVRŽENÉ ŠIFRY

V této kapitole bude rozebrána rychlost a výpočetní složitost navržené šifry a jejich součástí. Bude provedeno testování a posouzení rychlosti a výpočetní složitosti jednotlivých operací navržené šifry v závislosti na velikosti souboru (počtu šifrovaných bloků) a obsahu vstupních dat. Dále bude porovnávána rychlost navržené šifry s rychlostí šifer AES, DES a 3DES.

Před začátkem testování je nutné říci, že navržená šifra je implementována v programovacím jazyce Python verze 3.x, stejně jako šifry AES, DES a 3DES. Dále, že šifra byla navržená s ohledem zejména na bezpečnost, ne na rychlost. Test rychlosti může ovlivnit spoustu faktorů, například:

- zdrojový kód (kvalita samotné implementace) šifer
- vytížení počítače
- různé operace spojené s režii při běhu programu, které provádí prostředí Pythonu
- implementace operací jazyka Python (operace modulo, načtení prvku z pole, apod.)

Pro všechna testování byly vygenerované náhodné data. Měření času probíhalo pomocí knihovny *time*, která je součástí jazyka Python verze 3.x. Pro otestování šifer DES, 3DES a AES je použita knihovna kryptografických nástrojů pro Python verze 3.x nejnovější verze PyCrypto 2.6.1. A všechno testování proběhlo na notebooku s následující konfigurací:

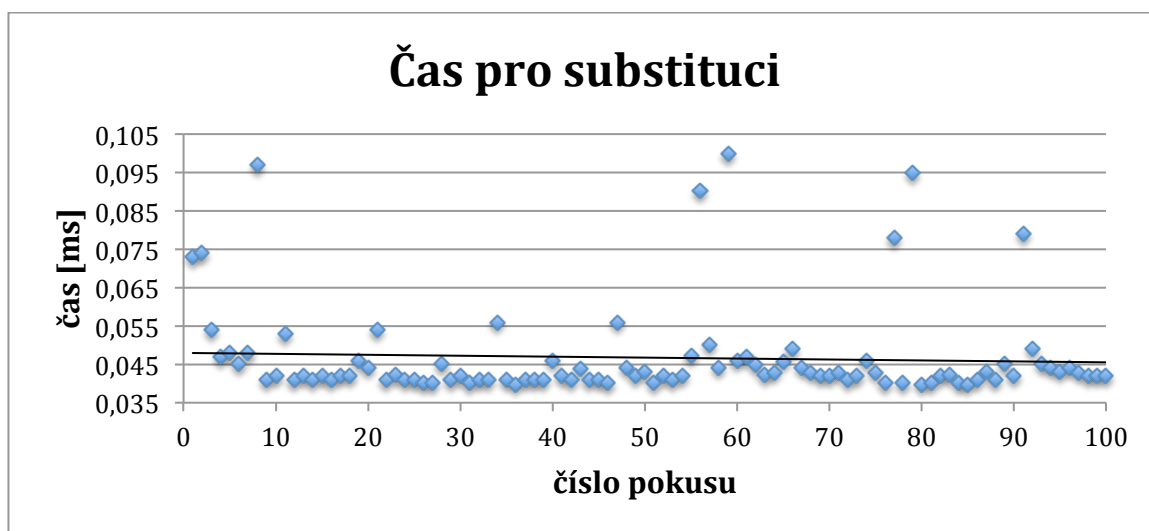
- MacBook Air 11“
- Procesor – Intel Core2Duo i7 2 GHz
- Paměť – 8GB DDR3 1600MHz
- Operační systém – Mac OS X 10.9.3
- Harddisk - APPLE SSD TS128E Media

4.1 Výpočetní složitosti a doby trvání operací pro šifrování

V této části odvodíme výpočetní složitost pro operace, které jsou použity k šifrování, a výpočetní složitost v závislosti na počtu vstupních bloků a počtu rund. Dále určíme jejich průměrnou dobu trvání pro náhodný vstup a náhodný klíč. Výpočetní složitost budeme značit velkým písmenem O a bude vztažena na operace při průchodu cyklem *for*. Takže výpočetní složitost značená jako $O(n)$ bude závislá na n průchodech cyklu *for*.

4.1.1 Výpočetní složitost a doba trvání operace substituce

Tato operace byla podrobně popsána v kapitole 3.4.1. Během substituce se prochází bajt za bajtem klíč a vstupní blok. Celkem 64 operací, kdy jako operaci bereme 1 průchod cyklem *for*. Pro obecnou velikost vstupního bloku dat je výpočetní složitost operace lineární - $O(n)$, kdy n je velikost bloku v bajtech. To ovšem platí za předpokladu, že bude konstantní operace pro načtení(uložení) prvku z(do) pole \rightarrow doba načtení(uložení) prvku na pozici i , kdy i je na intervalu $0..63$. Tím pádem se dá říci, že jediné rychlostní rozdíly v době šifrování substitucí mezi bloky s jiným obsahem, budou zapříčiněné chováním jazyka Python. Graf ukazující toto chování vypadá následovně (odchyly způsobené vytížením a chodem programu pro výpočet průměrné doby zanedbáme).

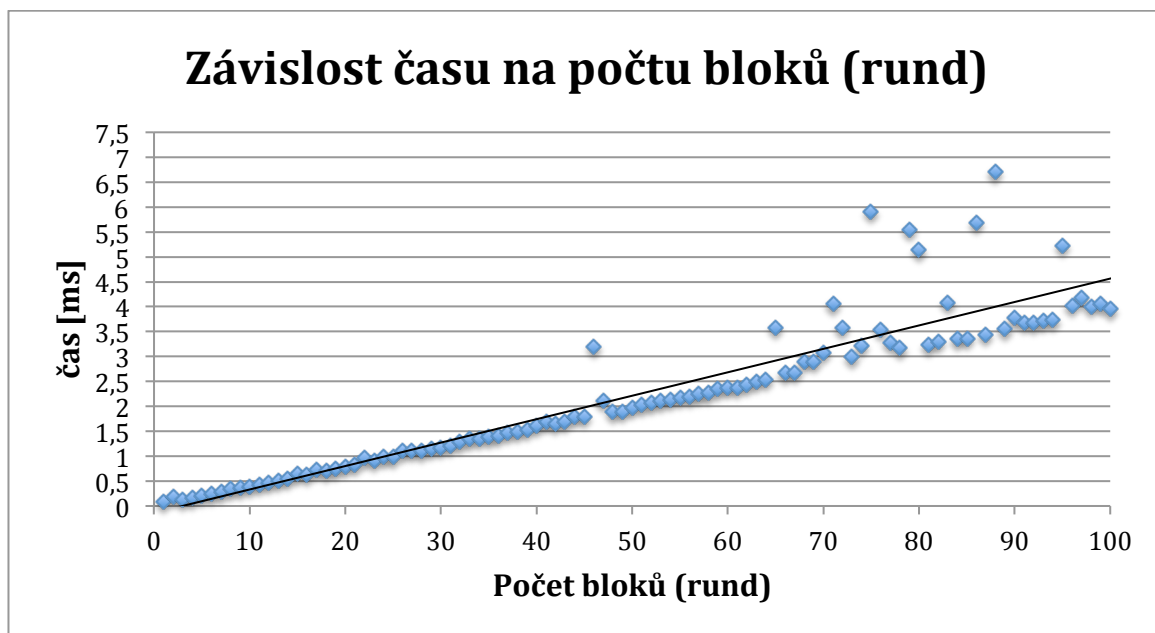


Obrázek 44 – Graf času při substituci pro náhodně generovaný vstupní blok a klíč

Rychlost substituce byla otestována stokrát, kdy do každého testu vstupoval 1 náhodně generovaný blok vstupních dat a náhodně generovaný klíč. Spojnice trendu má až na drobné odchylky konstantní průběh, takže můžeme říci, že operace má v našem případě konstantní trvání pro jakýkoliv vstupující blok do šifrování a jakýkoliv klíč a výpočetní složitost substituce je konstantní - $O(64)$. Ony odchylky nastávají v důsledku nekonstantní doby trvání načtení prvku z pole a dalších režií jazyka Python. Dále byla vypočítaná průměrná doba trvání šifrování 1 bloku dat pomocí substituce, na výše uvedené konfiguraci hardwaru, přibližně 0,041ms (viz. graf výše).

Výpočetní složitost šifrování pro b bloků, r rund a velikost bloku n v bajtech je lineární - $O(b \cdot r \cdot n)$. To znamená, že celková doba trvání substituce bude ovlivněna počtem bloků pro šifrování a počtem rund. Tím pádem, pokud vezmeme soubor o velikosti 640 bajtů (10 bloků), bude doba šifrování průměrně 0,41ms, protože se vždy šifruje jen jeden

blok dat, nepracuje se s celým souborem jako jedním řetězcem. Stejně tak, pokud budeme šifrovat jeden blok vstupních dat (64 bajtů) v 10 rundách, bude doba trvání substituce průměrně 0,41ms. Runda znamená počet substitucí, proto čas doby trvání 10 rund bude jako doba pro šifrování 10 bloků substitucí. Testování proběhlo pro konstantní počet rund – 1. Kdyby narůstal i počet rund, křivka trendu by rostla přímo úměrně počtu rund. Závislost ukazuje následující graf.



Obrázek 45 - Graf závislosti času pro substituci na počtu vstupních bloků (rund)

Jak můžeme vidět v grafu, tak pro vstupní data o velikosti 640 bajtů (100 bloků) dostaneme čas přibližně 4,06ms, což odpovídá předpokladu lineární složitosti - $O(b \cdot r \cdot n)$, kdy b je počet vstupních bloků, r je počet rund a n je velikost vstupního bloku v bajtech. Doba šifrování 1 bloku substitucí byla průměrně 0,041ms a tady vidíme, že pro 100 bloků je to přibližně stonásobek. Jelikož jsme si řekli, že šifrování 2 bloků je stejné jako šifrování jednoho bloku dvěma rundami, tak pro výpočet doby trvání substituce při šifrování, použijeme následující obecný vztah:

$$t = k \cdot b \cdot r \text{ [ms]} \quad (72)$$

- t – celkový čas v milisekundách
- b – počet vstupních bloků k zašifrování
- r – počet rund
- k – konstanta udávající průměrný čas potřebný k zašifrování jednoho bloku v jedné rundě, závislá na hardwarové konfiguraci, pro operaci substituce

V našem případě

$$t = 0,041 \cdot b \cdot r \text{ [ms]} \quad (73)$$

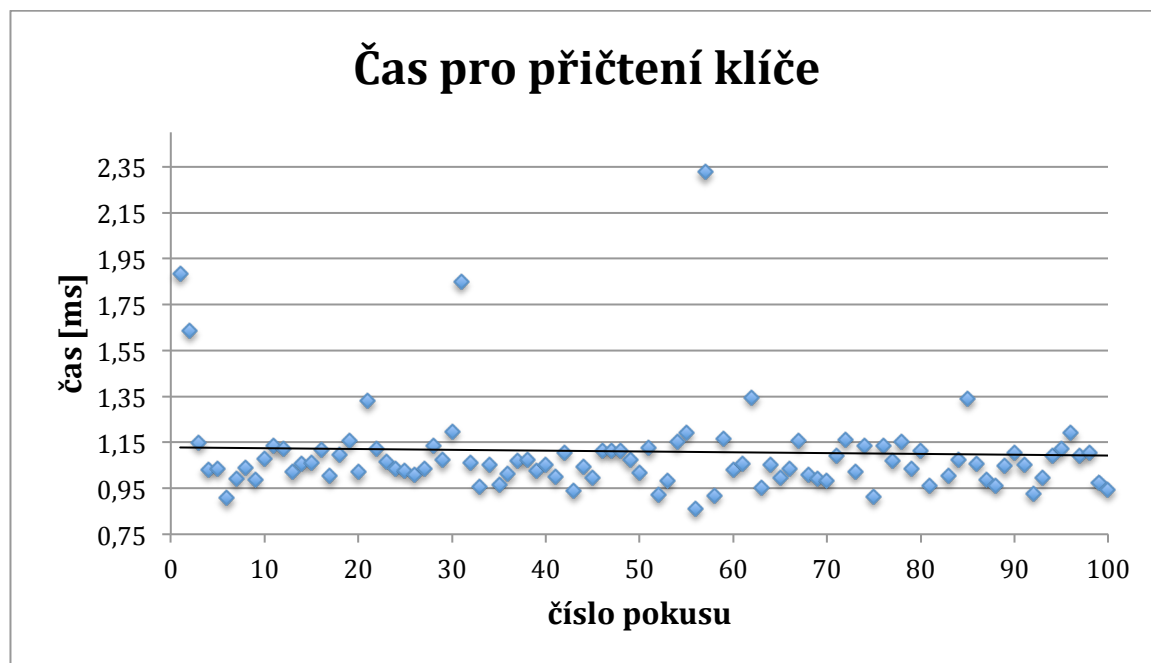
4.1.2 Výpočetní složitost a doba trvání operace přičtení klíče – Add

Nyní si odvodíme výpočetní složitost operace pro šifrování přičtení klíče – Add. Operace byla podrobně popsána v kapitole 3.4.2. Z výše uvedeného popisu můžeme říci, že nyní už doba trvání závisí na klíči. Jako jednu operaci uvažujeme jeden průchod vnitřním cyklem (přičtení hodnoty jednoho bajtu klíče k jedné hodnotě bajtu vstupního bloku dat). Počet přičtení závisí na klíči. Pokud by nastal nejhorší případ, tak se bude hodnota bajtu klíče přičítat vždy k nejvíce bajtům vstupu \rightarrow pro pozici 0 přičítání doprava (64 přičtení), pro pozici 1 přičítání doprava (63 přičtení) až po pozici $n/2-1$ (31), kdy n je velikost bloku (64), přičítání doprava (33 přičtení). Potom pro pozici $n/2$ až $n-1$ naopak přičítání doleva (33 až 64 přičtení). Celkem tedy maximálně $2 \cdot (64+63+\dots+34+33)$ nebo $2 \cdot (n+(n-1)+\dots+(n/2+2)+(n/2+1))$ přičtení (průchodů vnitřním cyklem for), kdy n je velikost bloku dat (klíče). Samozřejmě může nastat opačný případ, ale pro výpočetní složitost budeme předpokládat nejhorší případ. Z výše uvedeného můžeme odvodit následující výpočetní složitost pro operace přičtení modulo – Add, která je

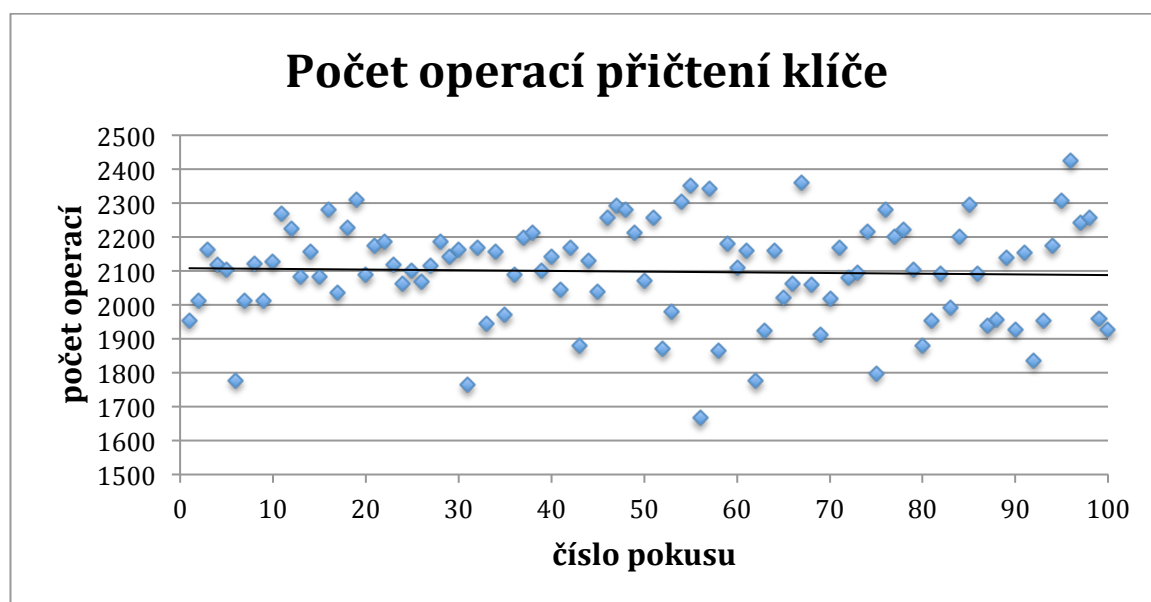
$$O\left(2 \cdot \left(\frac{\frac{n}{2} \cdot \left(n + \frac{n}{2} + 1\right)}{2}\right)\right) = O\left(\frac{3}{4} \cdot n^2 + \frac{n}{2}\right) \quad (74)$$

Výsledná výpočetní složitost pro operace přičtení klíče ke vstupní datům – Add je $O(3n^2/4+n/2)$, tedy kvadratická k velikosti bloku. V našem případě maximálně $(3 \cdot (64^2)/4+64/2) = 3104$ operací přičtení klíče ke vstupním datům na 1 vstupní blok dat, proto si můžeme označit výpočetní složitost za konstantní $\rightarrow O(3104)$. Při 100 testování operace přičtení klíče pro 100 náhodně generovaných vstupů o velikosti 64 bajtů a pro 100 náhodně generovaných klíčů při výše uvedené konfiguraci hardwaru nám vyšla následující průměr-

ná doba trvání 1,12ms a průměrný počet operací 2098 → Průměrné 2098/64 víc operací než substituce, což jí dělá přibližně 32,7 krát pomalejší (protože na 1 bajt připadá průměrně 33 operací). Pokud vydělíme $1,12/32,7$ dostaneme přibližně 0,034ms, což je cca doba trvání šifrování substitucí (0,041ms). V nejhorším případě by byla $3104/64 = 48,5$ krát pomalejší. Výsledek ukazují následující grafy (odchytky způsobené vytížením a chodem programu pro výpočet průměrné doby zanedbáme).

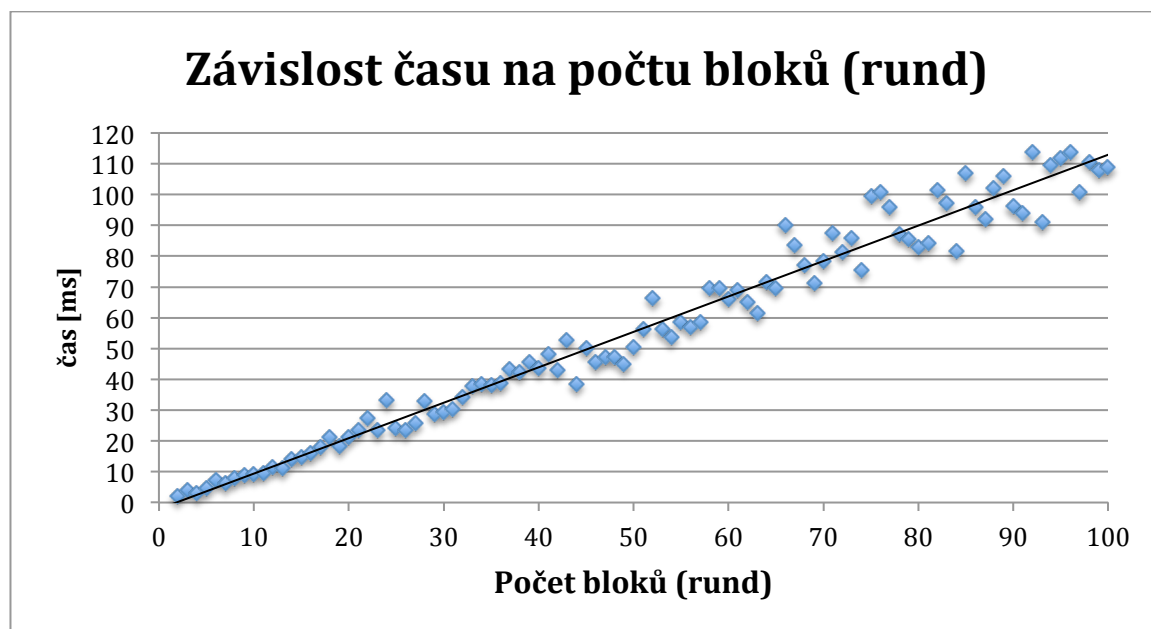


Obrázek 46 - Graf času pro přičtení klíče pro náhodně generovaný klíč



Obrázek 47 – Graf počtu operací přičtení klíče pro náhodně generovaný klíč

Výpočetní složitost v závislosti na počtu bloků b k dešifrování, počtu rund r a velikosti bloku dat n při přičítání klíče je opět lineární - $O(b \cdot r \cdot (3 \cdot n^2/4 + n/2))$. Testování proběhlo pro konstantní počet rund – 1. Kdyby narůstal i počet rund, křivka trendu by rostla přímo úměrně počtu rund. Jak ukazuje následující graf.



Obrázek 48 - Graf závislosti času pro přičtení klíče na počtu vstupních bloků (rund)

Pro 100 bloků šifrovaných pomocí přičtení klíče máme čas přibližně 107,9ms, což odpovídá přibližně stonásobku času pro průměrnou dobu trvání šifrování jednoho bloku operací přičtení klíče, zjištěné výše. Kolísání je způsobeno použitím jiného klíče. Tudiž, pro dobu trvání šifrování vstupních dat o b blocích a r rundách platí následující vztah.

$$t = k \cdot b \cdot r \text{ [ms]} \quad (75)$$

- t – celkový čas v milisekundách
- b – počet vstupních bloků k zašifrování
- r – počet rund
- k – konstanta udávající průměrný čas potřebný k zašifrování jednoho bloku v jedné rundě, závislá na hardwarové konfiguraci, pro operaci přičtení klíče

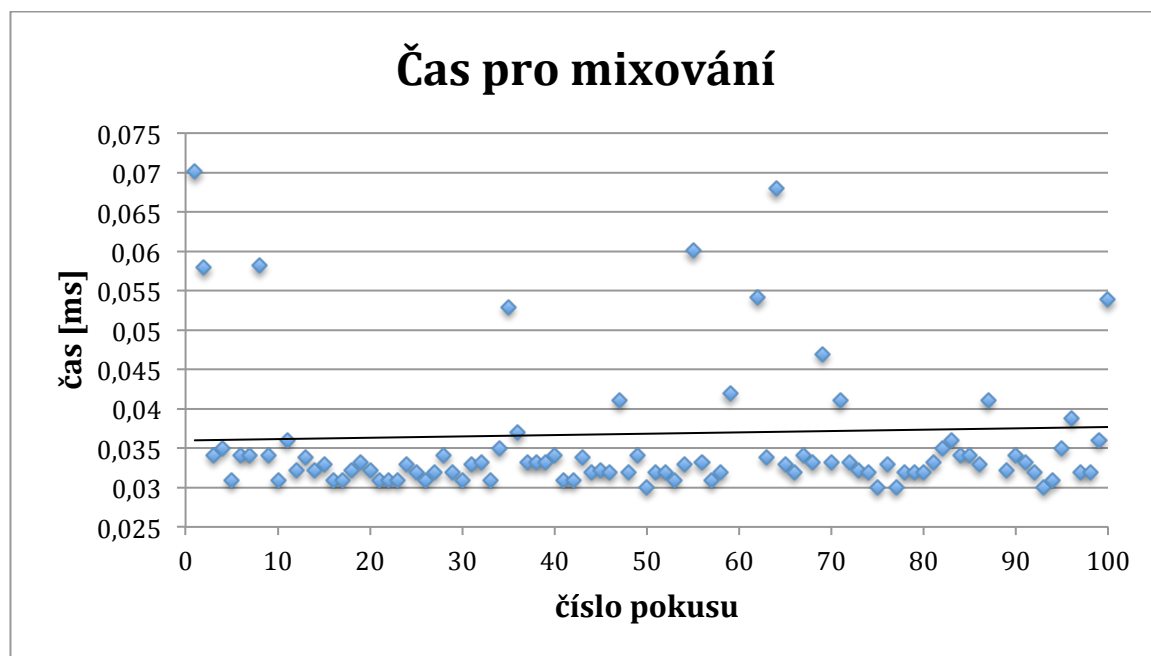
V našem případě

$$t = 1,12 \cdot b \cdot r \text{ [ms]} \quad (76)$$

4.1.3 Výpočetní složitost a doba trvání operace mixování

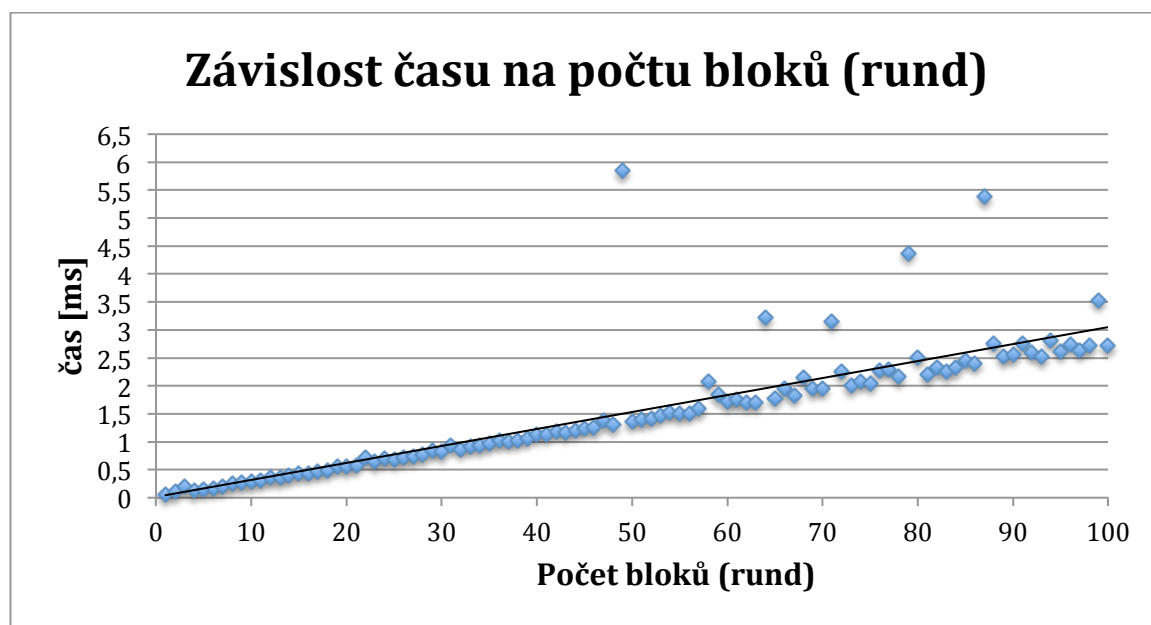
Operace byla podrobně popsána v kapitole 3.4.3. Pro získání výpočetní složitosti operace pro šifrování mixování (transpozice), můžeme vycházet z výpočetní složitosti pro substituci, protože opět na 1 bajt vstupních dat připadá jeden průchod cyklem *for* (1 operace). Tedy pro zašifrování jednoho bloku (64 bajtů) budeme mít konstantní výpočetní složitost $O(64)$, kdy 64 je velikost vstupního bloku v bajtech. Obecně má složitost lineární k velikost vstupního bloku $\rightarrow O(n)$

Doba trvání šifrování jednoho bloku vstupních dat pomocí mixování by nám měl vyjít zhruba jako doba trvání šifrování jednoho bloku vstupních dat substitucí. Uděláme stejný test jako pro substituci - otestujeme mixování pro náhodně generovaný vstupní blok a náhodně generovaný klíč 100 krát, a pak vypočítáme průměrnou dobu trvání šifrování vstupního bloku pomocí operace mixování. Z testování nám vyšla průměrná doba šifrování jednoho bloku vstupních dat 0,032ms, což je opravdu velmi podobné době trvání šifrování vstupního bloku dat substitucí. Testování zobrazuje následující graf (odchyly způsobené vytížením a chodem programu pro výpočet průměrné doby zanedbáme).



Obrázek 49 - Graf času při mixování pro náhodně generovaný vstupní blok a klíč

Výpočetní složitost v závislosti na počtu bloků b , počtu rund r a velikosti bloku dat n při mixování je opět lineární - $O(b \cdot r \cdot n)$. Testování proběhlo pro konstantní počet rund – 1. Kdyby narůstal i počet rund, křivka trendu by rostla přímo úměrně počtu rund. Jak ukazuje následující graf.



Obrázek 50 - Graf závislosti času pro mixování na počtu vstupních bloků (rund)

Tudíž, pro dobu trvání šifrování vstupních dat o b blocích a r rundách platí následující vztah.

$$t = k \cdot b \cdot r \text{ [ms]} \quad (77)$$

- t – celkový čas v milisekundách
- b – počet vstupních bloků k zašifrování
- r – počet rund
- k – konstanta udávající průměrný čas potřebný k zašifrování jednoho bloku v jedné rundě, závislá na hardwarové konfiguraci, pro operaci mixování

V našem případě

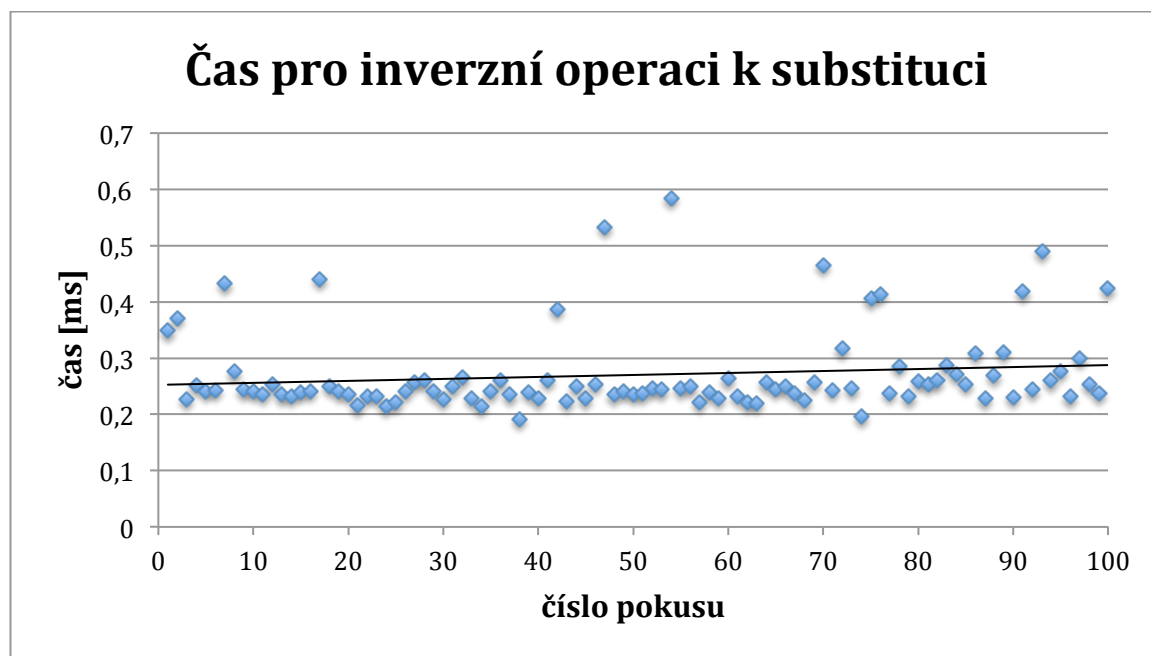
$$t = 0,032 \cdot b \cdot r \text{ [ms]} \quad (78)$$

4.2 Výpočetní složitosti a doby trvání operací pro dešifrování

V této části odvodíme výpočetní složitost pro inverzní operace k operacím pro šifrování, které jsou použity k dešifrování, a výpočetní složitost v závislosti na počtu vstupních bloků k dešifrování a počtu rund. Dále určíme jejich průměrnou dobu trvání pro náhodný vstup a náhodný klíč.

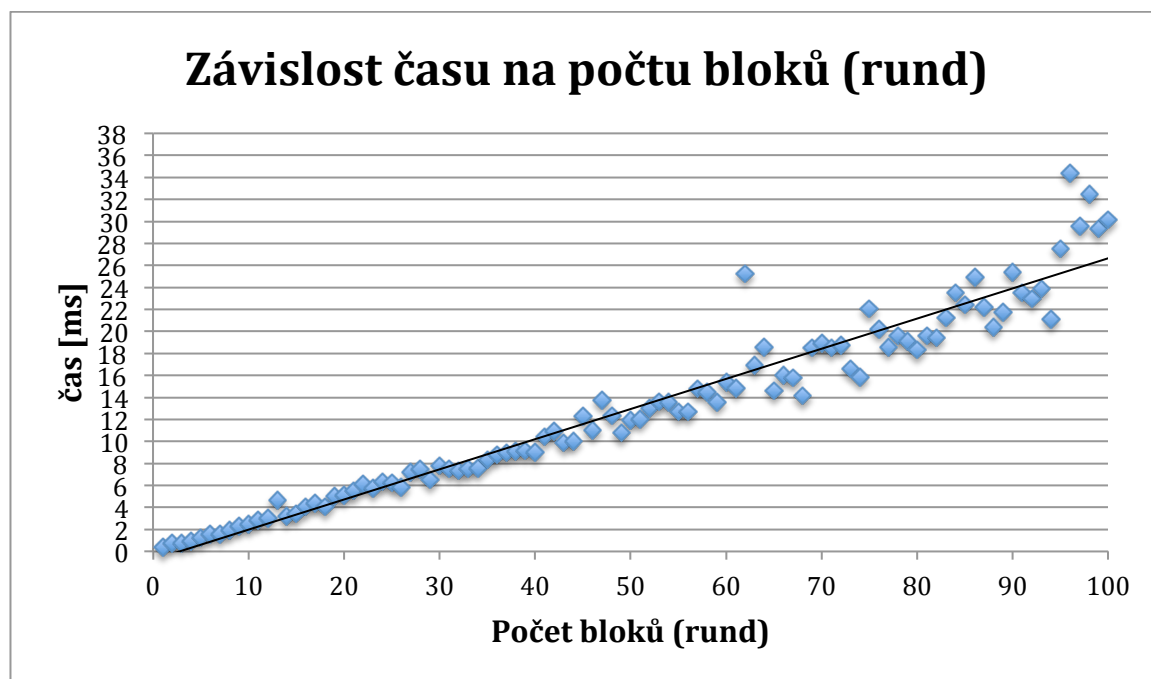
4.2.1 Výpočetní složitost a doba trvání „DeSubstituce“ – inverzní operace k substituci

Operace byla podrobně popsána v kapitole 3.5.1. Pro inverzní operaci k substituci platí, že pro dešifrování každého vstupního bajtu dat se provede vyhledávání v poli, což znamená až n operací, kdy n je velikost bloku vstupních zašifrovaných dat. To v nejhorším případě znamená $64 \cdot 64 = 4096$ operací při inverzní operaci k substituci. Tento fakt nám dává výpočetní složitost $O(n^2)$, kdy n je velikost bloku. V našem případě, kdy se velikost bloku nemění, můžeme brát výpočetní složitost za konstantní $\rightarrow O(4096)$. Dešifrování jednoho bloku zašifrovaných dat může trvat až 64 krát déle, než tomu bylo při šifrování bloku dat. Doba trvání operace je závislá na datech k dešifrování. Pro demonstraci tohoto faktu opět provedme 100 krát dešifrování pro zašifrované náhodné bloky a pro náhodné klíče. Z testu jsme zjistili průměrnou dobu trvání inverzní operace k substituci 0,27ms a výsledek testu znázorňuje tento graf (odchyly způsobené vytížením a chodem programu pro výpočet průměrné doby zanedbáme).



Obrázek 51 - Graf času při inverzní operaci k substituci pro náhodně generovaný vstup a klíč

Výpočetní složitost v závislosti na počtu bloků b k dešifrování, počtu rund r a velikosti bloku dat n při inverzní operaci k substituci je opět lineární - $O(b \cdot r \cdot n)$. Testování proběhlo pro konstantní počet rund – 1. Kdyby narůstal i počet rund, křivka trendu by rostla přímo úměrně počtu rund. Jak ukazuje následující graf.



Obrázek 52 - Graf závislosti času pro inverzní operaci k substituci na počtu vstupních bloků (rund)

Tudíž, pro dobu trvání dešifrování vstupních dat o b blocích a r rundách platí následující vztah.

$$t = k \cdot b \cdot r \text{ [ms]} \quad (79)$$

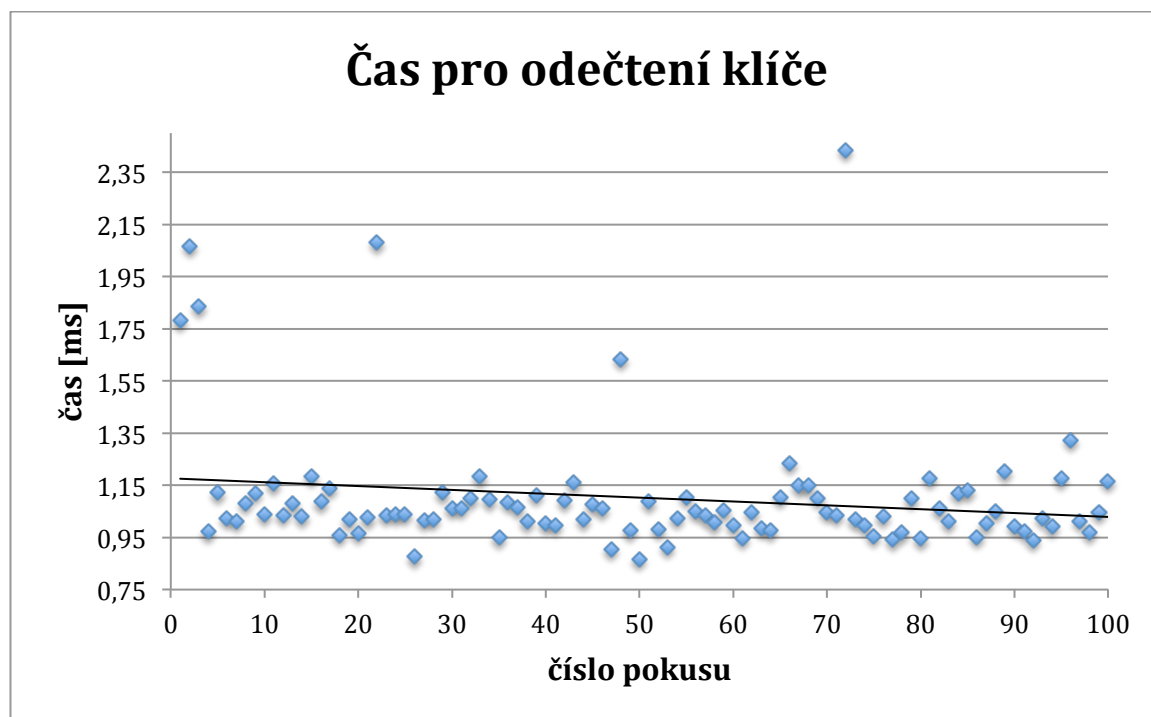
- t – celkový čas v milisekundách
- b – počet vstupních bloků k dešifrování
- r – počet rund
- k – konstanta udávající průměrný čas potřebný k dešifrování jednoho bloku v jedné rundě, závislá na hardwarové konfiguraci, pro inverzní operaci k substituci

V našem případě

$$t = 0,27 \cdot b \cdot r \text{ [ms]} \quad (80)$$

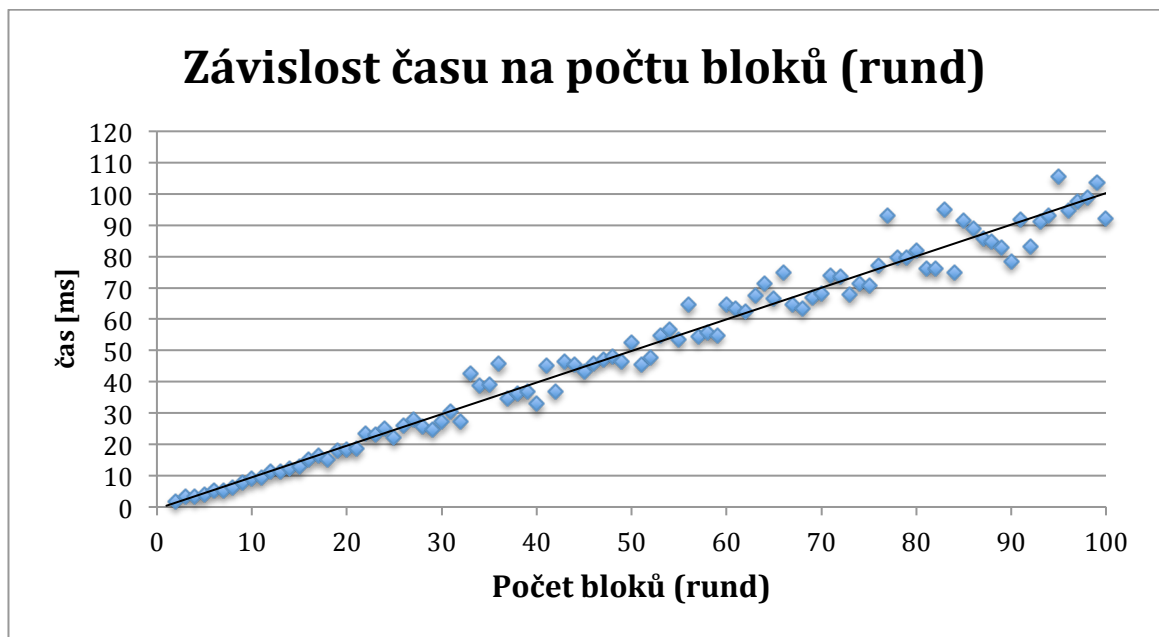
4.2.2 Výpočetní složitost a doba trvání „DeAdd“ – operace odečtení klíče od dat

Operace byla podrobně popsána v kapitole 3.5.2. Protože se operace odečtení klíče, až na průchod vstupního bloku zprava doleva namísto zleva doprava a odečítáním klíče namísto přičítání klíče, ničím jiným od operace Add – přičtení klíče neliší. Bude mít operace stejnou kvadratickou složitost - $O(3 \cdot n^2/4 + n/2)$, kdy n je velikost bloku. V našem případě při konstantní velikosti bloku, můžeme brát výpočetní složitost jako konstantní $\rightarrow O(3104)$. Pro zjištění průměrné doby trvání operace odečtení klíče opět provedme 100 krát dešifrování pro zašifrované náhodné bloky a pro náhodné klíče. Z testu jsme získali průměrnou dobu trvání odečtení klíče 1,1ms, což je až na 2 setiny stejně jako operace přičtení klíče. Výsledek je patrný z následujícího grafu (odchyly způsobené vytížením a chodem programu pro výpočet průměrné doby zanedbáme).



Obrázek 53 - Graf času při odečtení klíče pro náhodné data a klíč

Výpočetní složitost v závislosti na počtu bloků b k dešifrování, počtu rund r a velikosti bloku dat n při odečítání klíče je opět lineární - $O(b \cdot r \cdot (3 \cdot n^2/4 + n/2))$. Testování proběhlo pro konstantní počet rund – 1. Kdyby narůstal i počet rund, křivka trendu by rostla přímo úměrně počtu rund. Jak ukazuje následující graf.



Obrázek 54 - Graf závislosti času pro odečtení klíče na počtu vstupních bloků (rund)

Tudíž, pro dobu trvání dešifrování vstupních dat o b blocích a r rundách platí následující vztah.

$$t = k \cdot b \cdot r \text{ [ms]} \quad (81)$$

- t – celkový čas v milisekundách
- b – počet vstupních bloků k dešifrování
- r – počet rund
- k – konstanta udávající průměrný čas potřebný k dešifrování jednoho bloku v jedné rundě, závislá na hardwarové konfiguraci, pro odečtení klíče

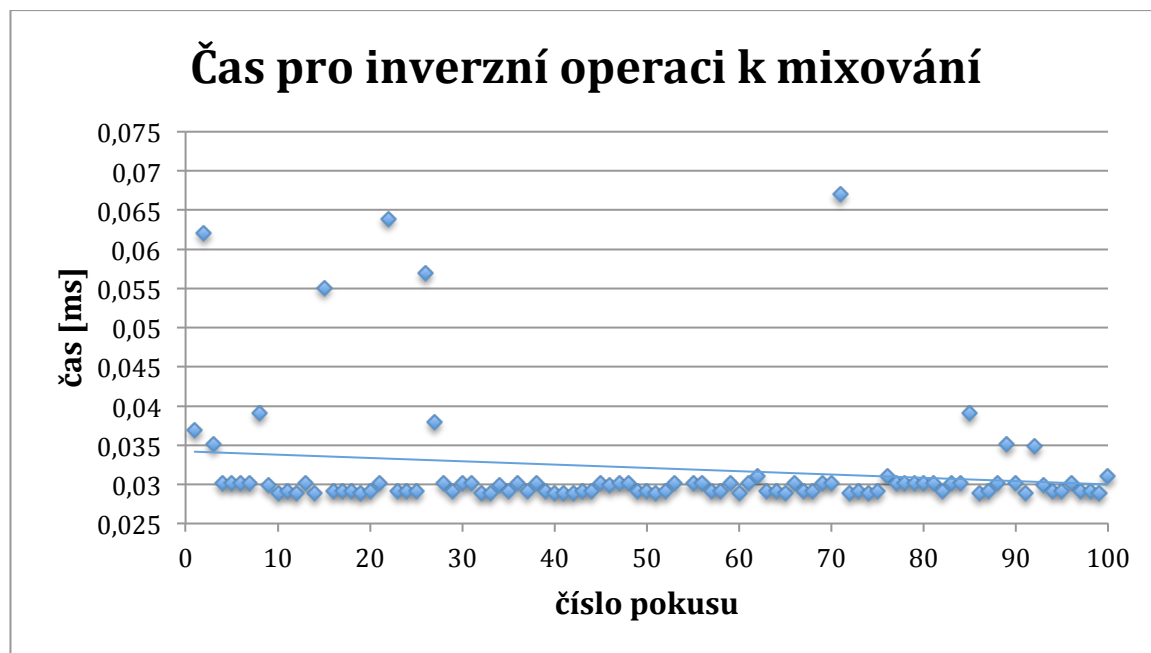
V našem případě

$$t = 1,1 \cdot b \cdot r \text{ [ms]} \quad (82)$$

4.2.3 Výpočetní složitost a doba trvání „DeMixování“ – inverzní operace k mixování

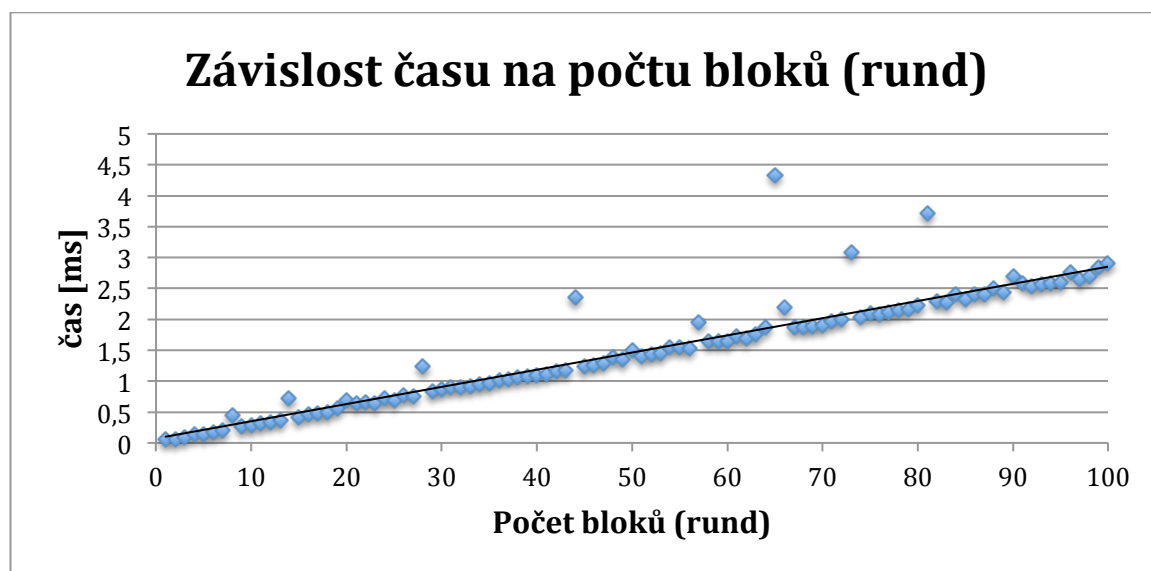
Operace byla podrobně popsána v kapitole 3.5.3. Protože se inverzní operace k mixování, až na průchod vstupního bloku zprava doleva namísto zleva doprava, ničím jiným od operace mixování neliší. Bude mít operace stejnou lineární - $O(n)$, n je velikost bloku. V našem případě ale bude výpočetní složitost konstantní $O(64)$ Pro zjištění průměrné doby trvání operace odečtení klíče opět provedme 100 krát dešifrování pro zašifrované náhodné bloky a pro náhodné klíče. Z testu jsme získali průměrnou dobu trvání inverzní operace

k mixování 0,028ms, což je až na 1 setiny stejně jako operace mixování při šifrování. Výsledek je patrný z následujícího grafu (odchylky způsobené vytížením a chodem programu pro výpočet průměrné doby zanedbáme).



Obrázek 55 - Graf času při inverzní operaci k mixování pro náhodný vstup

Výpočetní složitost v závislosti na počtu bloků b k dešifrování, počtu rund r a velikosti bloku dat n při inverzní operaci k mixování je opět lineární - $O(b \cdot r \cdot n)$. Testování proběhlo pro konstantní počet rund – 1. Kdyby narůstal i počet rund, křivka trendu by rostla přímo úměrně počtu rund. Jak ukazuje následující graf.



Obrázek 56 - Graf závislosti času pro inverzní operaci k mixování na počtu vstupních bloků (rund)

Tudíž, pro dobu trvání dešifrování vstupních dat o b blocích a r rundách platí následující vztah.

$$t = k \cdot b \cdot r \text{ [ms]} \quad (83)$$

- t – celkový čas v milisekundách
- b – počet vstupních bloků k dešifrování
- r – počet rund
- k – konstanta udávající průměrný čas potřebný k dešifrování jednoho bloku v jedné rundě, závislá na hardwarové konfiguraci, pro inverzní funkci k mixování

V našem případě

$$t = 0,028 \cdot b \cdot r \text{ [ms]} \quad (84)$$

4.3 Srovnání operací pro šifrování a dešifrování

Srovnání výsledných dob pro zašifrování/dešifrování jednoho bloku dat a odvozené výpočetní složitosti ukazuje následující tabulka.

Tabulka 3 - Tabulka srovnání operací pro šifrování a dešifrování

Šifrování			
Název operace	Výpočetní složitost na n bajtů vstupních dat	Výpočetní složitost navržené šifry, kdy n = 64	Průměrná doba trvání [ms]/blok
Substituce	$O(n)$	$O(64)$	0,041
Přičtení klíče	$O(3n^2/4+n/2)$	$O(3104)$	1,12
Mixování	$O(n)$	$O(64)$	0,032
Celkově		$O(3232)$	1,193
Dešifrování			
Název operace	Výpočetní složitost na n bajtů vstupních dat	Výpočetní složitost navržené šifry, kdy n = 64	Průměrná doba trvání [ms]/blok
"DeSubstituce"	$O(n^2)$	$O(4096)$	0,27
Odečtení klíče	$O(3n^2/4+n/2)$	$O(3104)$	1,1
"DeMixování"	$O(n)$	$O(64)$	0,028
Celkově		$O(7264)$	1,398

Jak vyplývá z tabulky, tak průměrná doba trvání šifrování a dešifrování jednoho bloku dat se liší. Zejména díky inverzní operaci k substituci, která má vyšší výpočetní složitost. Průměrná doba dešifrování pro náhodně generovaný vstup je průměrně $1,398/1,193 = 1,172$ krát vyšší, tedy průměrně trvá o 17,2% delší dobu. To ovšem ani zdaleka není stejné jako pro nejhorší případ, kdy může až k více jak dvojnásobné době trvání dešifrování ku době trvání šifrování, ale tato situace je vysoce nepravděpodobná. Tudíž navržená šifra má asynchronní chování → doba pro šifrování je rozdílná od doby pro dešifrování.

Operace pro substituci a pro mixování mají sice stejnou výpočetní složitost, ale rozdíl v době jejich trvání je zapříčiněn dobou trvání 1 operace (trvání 1 průchodu cyklem *for*).

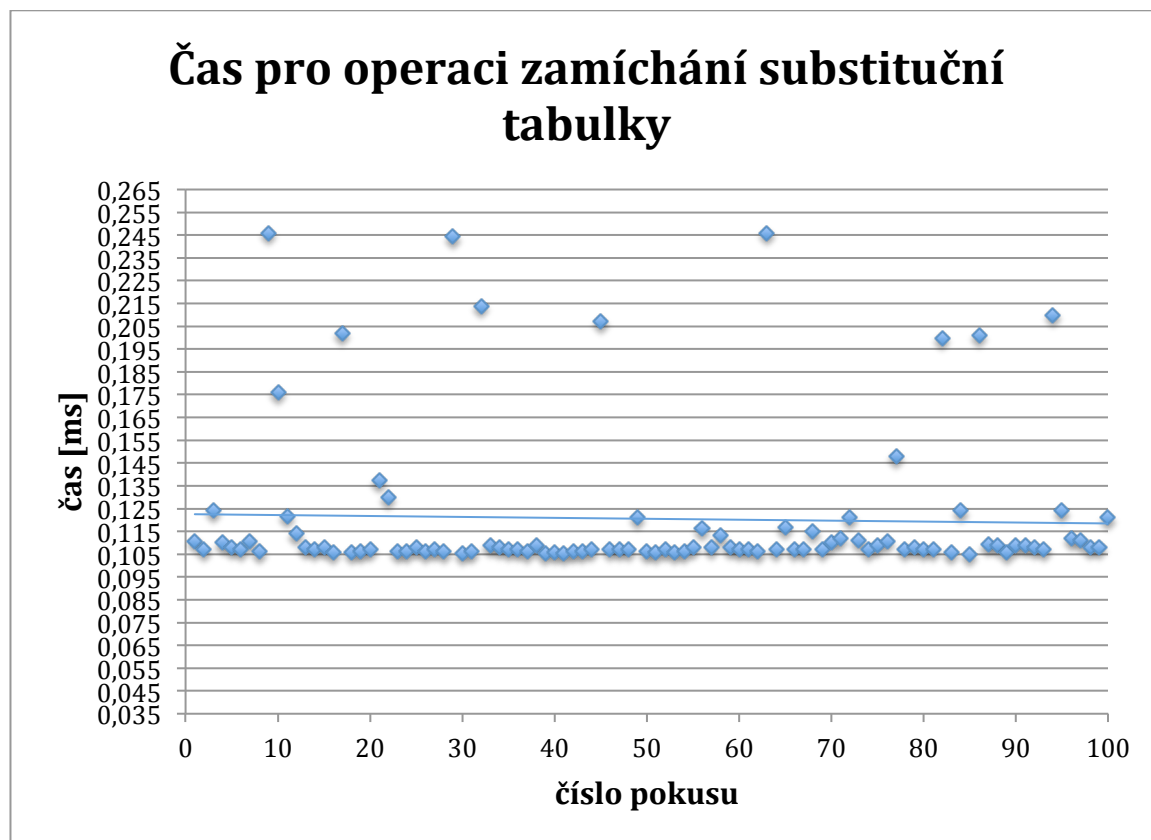
4.4 Výpočetní složitost a doby trvání operací pro režii

V této části odvodíme výpočetní složitost pro režijní operace, které jsou použity při šifrování/dešifrování, a výpočetní složitost v závislosti na vstupu těchto funkcí. Bude se jednat o operace - zamíchání substituční tabulky, operaci pro posunutí klíče, operaci pro vygenerování posloupnosti operací pro šifrování/dešifrování a operaci pro generování inicializačního vektoru.

4.4.1 Výpočetní složitost a doba trvání operace pro zamíchání substituční tabulky

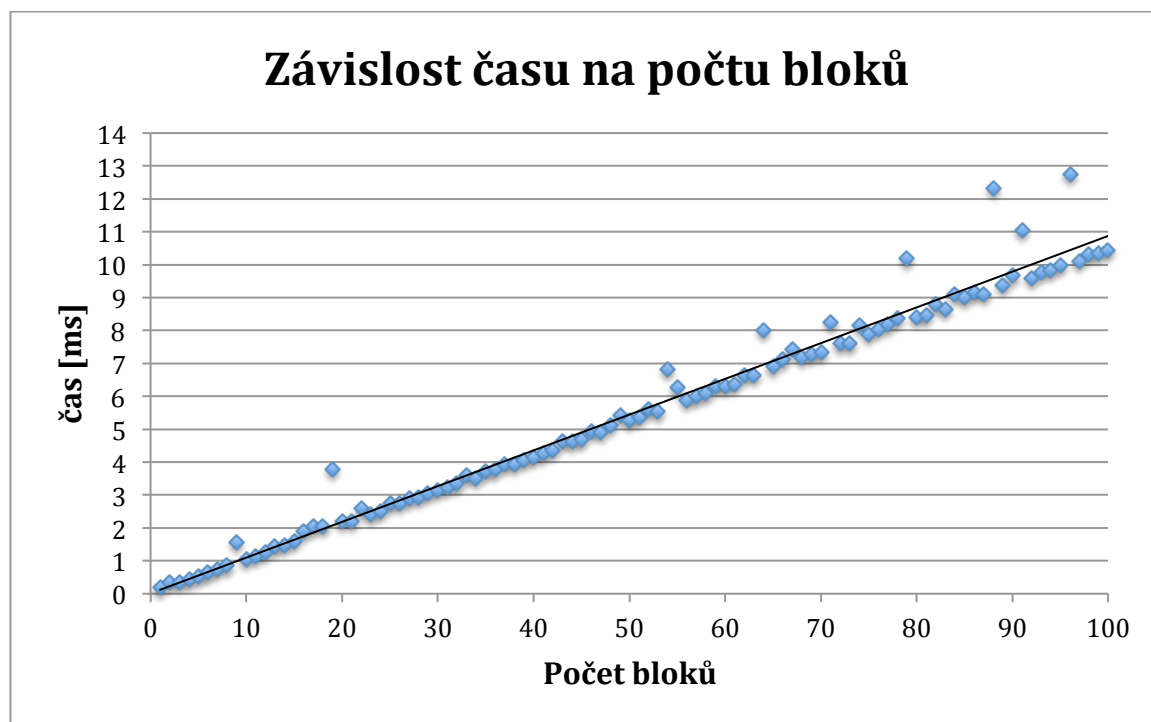
Operace je podrobně popsána v kapitole 3.3. Operace obsahuje pouze jeden cyklus *for* o konstantním počtu kroků 256. Takže má konstantní výpočetní složitost $O(256)$. Operace je výpočetně nezávislá na klíči, i když je klíčem řízená. Operace probíhá stejně jak pro šifrování, tak i pro dešifrování. Operace by měla být v průměru přibližně 4 krát náročnější jako operace pro šifrování mixování, která měla konstantní výpočetní složitost $O(64)$ a má skoro stejné složení operací pro cyklus *for*. Pro naši konfiguraci jsme určili průměrnou

dobu trvání této operace opět 100 krát pro 100 náhodně vygenerovaných klíčů. Průměrná doba trvání operace je 0,108ms, což potvrzuje předpoklad přibližně 4-násobné doby trvání, v porovnání s operací pro mixování. Výsledek ukazuje následující graf (odchyly způsobené vytížením a chodem programu pro výpočet průměrné doby zanedbáme).



Obrázek 57 - Grafu času pro operaci zamíchání substituční tabulky pro náhodně generovaný klíč

Počet provedení operace k zamíchání substituční tabulky je stejný jako počet bloků. Už není závislý na počtu rund, protože k zamíchání substituční tabulky dochází až po provedení šifrování/dešifrování v n rundách. Výpočetní složitost závislá na počtu šifrovaných/dešifrovaných bloků je lineární – $O(b \cdot 256)$, kdy b je počet bloků k šifrování/dešifrování. Jak ukazuje následující graf.



Obrázek 58 - Graf závislosti času pro operaci zamíchání substituční tabulky na počtu šifrovaných/dešifrovaných bloků

Tudíž, pro dobu trvání operace pro zamíchání substituční tabulky při šifrování/dešifrování b bloků platí následující vztah.

$$t = k \cdot b[\text{ms}] \quad (85)$$

- t – celkový čas v milisekundách
- b – počet vstupních bloků k šifrování/dešifrování
- k – konstanta udávající průměrný čas potřebný k zamíchání substituční tabulky pro jeden blok dat, závislá na hardwarové konfiguraci, při šifrování/dešifrování

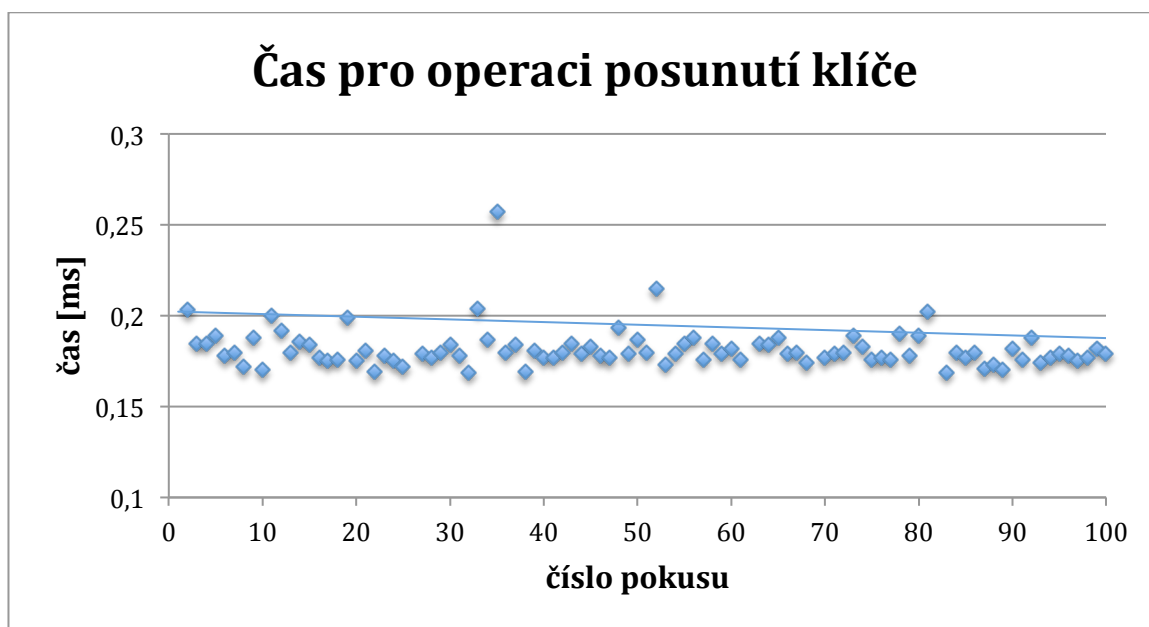
V našem případě

$$t = 0,108 \cdot b [\text{ms}] \quad (86)$$

4.4.2 Výpočetní složitost a doba trvání operace pro posunutí klíče

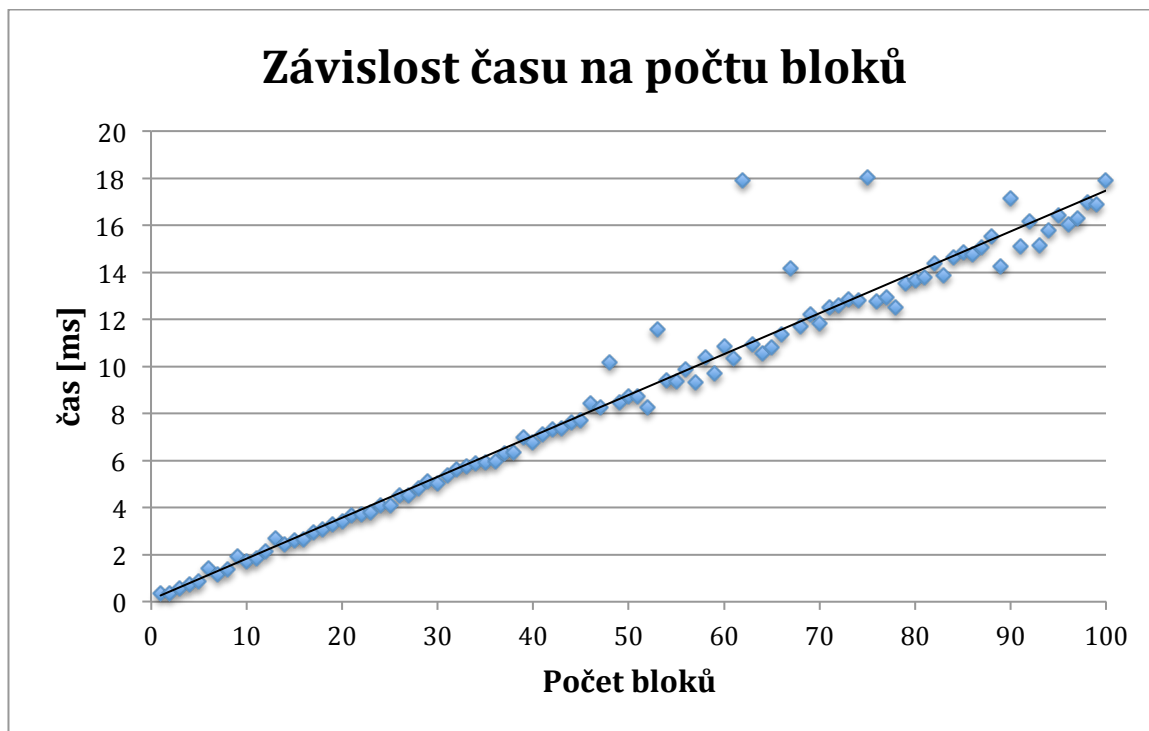
Operace byla podrobně rozebrána v kapitole 3.7. Co se týká výpočetní složitosti, pokud vezmeme jako jednu operaci průchod cyklem *for* tak se bude jednat o výpočetní složitost $O(n)$, kdy n je velikost bloku \rightarrow lineární k velikosti bloku, v našem případě konstantní $O(64)$, protože se velikost bloku nemění. Protože je ale tělo cyklu krapet složitější, trochu tuto výpočetní složitost zpřesníme. Průchod cyklem se ovlivněn ze součinu hodnoty bajtu klíče, hodnoty bajtu vstupních dat do šifrování/dešifrování a hodnoty bajtu výstup-

ních dat ze šifrování/dešifrování. Z toho důvodu si za 1 operaci označme výpočet rovnice v průchodu cyklem *for*, která je výpočetně nejnáročnější operací. Pokud vezmeme nejhorší případ, tak pro každý bajt nového klíče může dojít až ke 3 výpočtům rovnic na průchod cyklem. Tedy výpočetní složitost můžeme zpřesnit na $O(3n)$. Další zpřesnění bychom mohli docílit tím, že jako jednu operaci bychom vzali elementární operace v rámci každé rovnice (ščitání, odečítání, div, modulo, násobení), ale pro naše účely bude stačit přiblížení $O(3n)$, kdy n je velikost vstupního bloku dat (klíče). Pro naši šifru tedy přibližně $O(192)$. Průměrná doba trvání by se měla blížit průměrné době pro zamíchání substituční tabulky, ale může být i větší v závislosti na implementaci a počtu operací pro rovnice v jazyce Python. Pro výpočet průměrné doby trvání operace pro posunutí klíče jsme provedli 100 testování pro 100 náhodně generovaných vstupních dat. Průměrná doba nám testem vyšla na 0,179ms, což je přibližně jako doba pro zamíchání tabulky. Výsledek pokusu ukazuje následující graf (odchyly způsobené vytížením a chodem programu pro výpočet průměrné doby zanedbáme).



Obrázek 59 - Graf času pro operaci posunutí klíče pro náhodný vstup

Počet provedení operace posunutí klíče je stejný jako počet bloků. Není závislý na počtu rund, protože k posunutí klíče dochází až po provedení šifrování/dešifrování v n rundách. Výpočetní složitost závislá na počtu šifrovaných/dešifrovaných bloků je lineární – $O(b \cdot n)$, kdy b je počet bloků k šifrování/dešifrování a n je velikost bloku v bajtech. Jak ukazuje následující graf.



Obrázek 60 - Graf závislosti při posunutí klíče na počtu šifrovaných/dešifrovaných bloků

Tudíž, pro dobu trvání operace pro zamíchání substituční tabulky při šifrování/dešifrování b bloků platí následující vztah.

$$t = k \cdot b [\text{ms}] \quad (87)$$

- t – celkový čas v milisekundách
- b – počet vstupních bloků k šifrování/dešifrování
- k – konstanta udávající průměrný čas potřebný k posunutí klíče pro zašifrování/dešifrování jednoho bloku dat, závislá na hardwarové konfiguraci

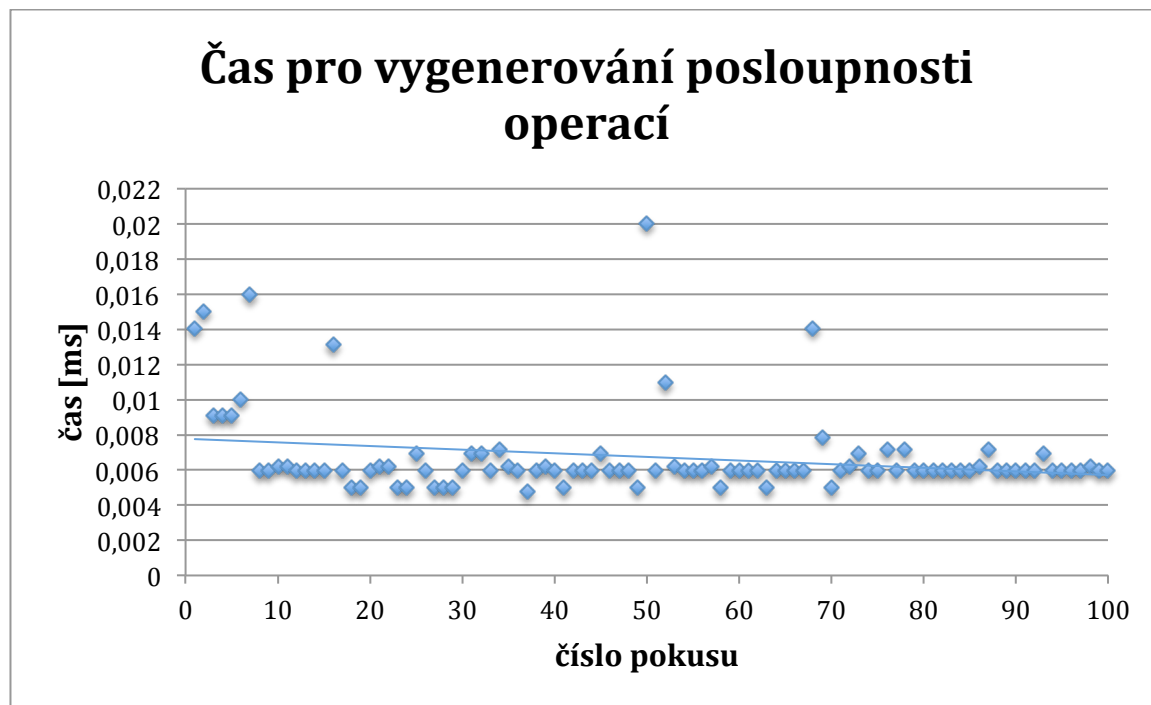
V našem případě

$$t = 0,179 \cdot b [\text{ms}] \quad (88)$$

4.4.3 Výpočetní složitost a doba trvání operace pro generování posloupnosti operací

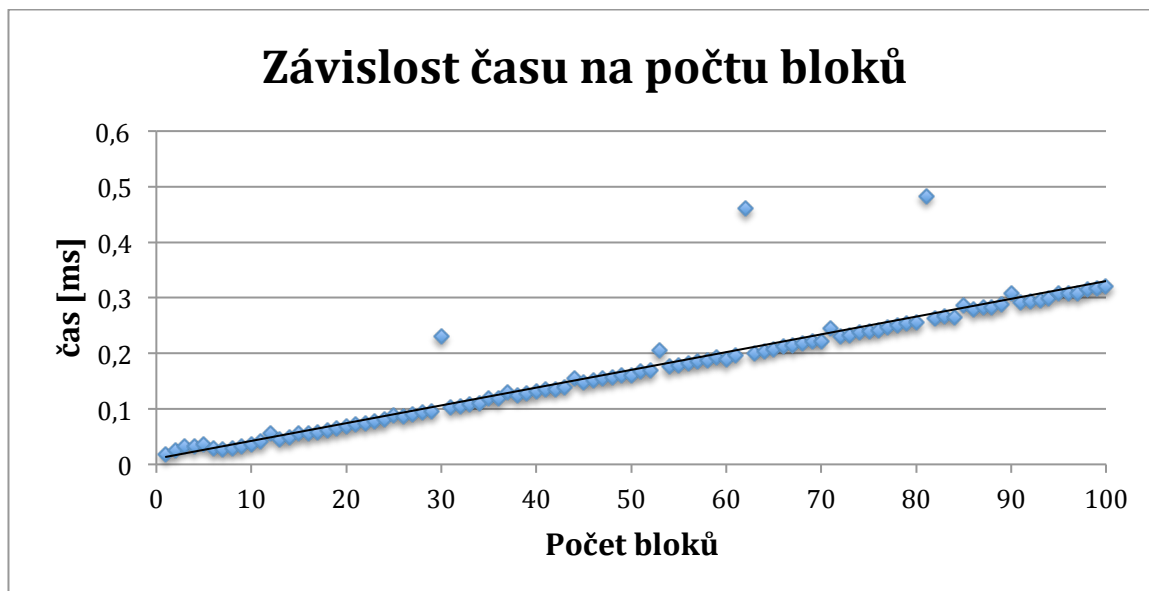
Operace byla podrobně rozebrána v kapitole 3.6. Co se týká výpočetní složitosti, odvodíme ji opět v závislosti na průchodech cyklů *for*. Operace obsahuje dva cykly *for*, kdy první je závislý na počtu rund a druhý je konstantní o 8 průchodech. Tudíž výsledná složitost nebude závislá na žádných vstupu (klíč a blok k zašifrování/dešifrování), ale pouze na počtu rund r . Proto je tedy výpočetní složitost lineární - $O(8r)$, kdy r je počet rund. Průměrná doba trvání vygenerování posloupnosti operací pro šifrování/dešifrování pro jednu rundu bude mít výpočetní složitost konstantní - $O(8)$. Pro výpočet průměrné do-

by trvání operace pro vygenerování posloupnosti operací pro jednu rundu jsme provedli 100 testování pro 100 náhodně generovaných vstupních dat (klíčů). Průměrná doba nám testem vyšla na 0,0059ms, což je přibližně jako osmina pro operaci substituce se složitostí $O(64)$. Výsledek pokusu ukazuje následující graf (odchyly způsobené vytížením a chodem programu pro výpočet průměrné doby zanedbáme).



Obrázek 61 - Grafu času pro vygenerování posloupnosti operací pro 1 rundu

Počet provedení operace posunutí klíče je stejný jako počet bloků, ale je také závislý na počtu rund, protože k vygenerování posloupnosti dochází před šifrováním/dešifrováním v r rundách a pro vygenerování se používá počet rund r . Výpočetní složitost závislá na počtu šifrovaných/dešifrovaných bloků je lineární – $O(b \cdot 8 \cdot r)$, kdy b je počet bloků k šifrování/dešifrování a r je počet rund. Testování proběhlo pro konstantní počet rund – 1. Kdyby narůstal i počet rund, křivka trendu by rostla přímo úměrně počtu rund. Jak ukazuje následující graf.



Obrázek 62 - Graf závislosti času pro vygenerování posloupnosti operací při šifrování/dešifrování na počtu bloků

Tudíž, pro dobu trvání operace pro zamíchání substituční tabulky při šifrování/dešifrování b bloků platí následující vztah.

$$t = k \cdot b \cdot r \text{ [ms]} \quad (89)$$

- t – celkový čas v milisekundách
- b – počet vstupních bloků k šifrování/dešifrování
- k – konstanta udávající průměrný čas potřebný vygenerování posloupnosti operací pro zašifrování/dešifrování jednoho bloku dat, závislá na hardwarové konfiguraci

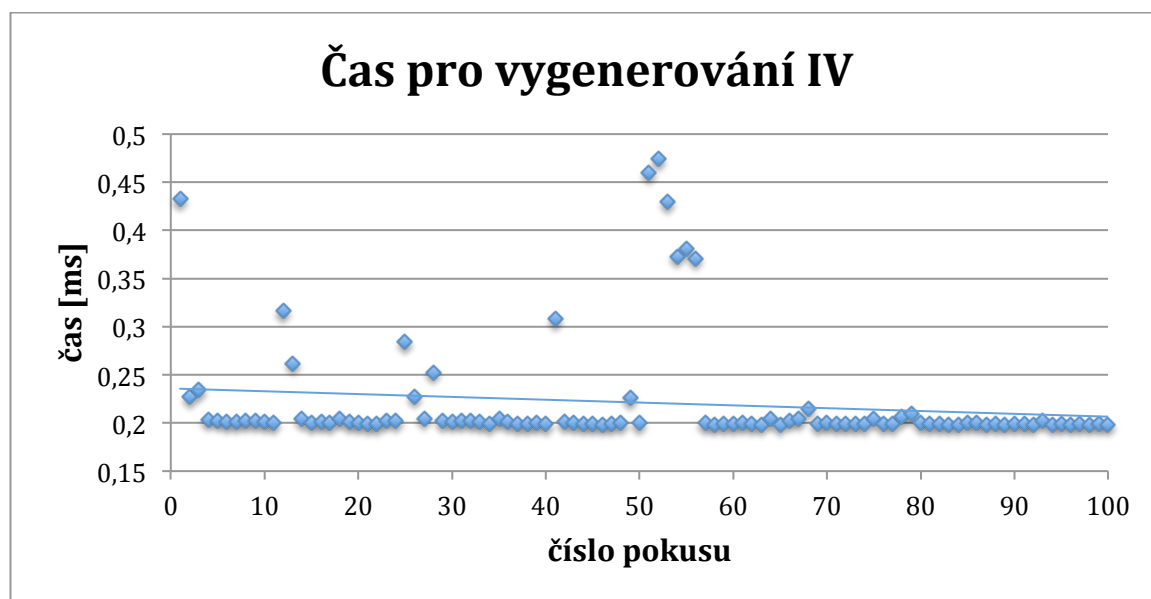
V našem případě

$$t = 0,0059 \cdot b \cdot r \text{ [ms]} \quad (90)$$

4.4.4 Výpočetní složitost a doba trvání operace generování IV

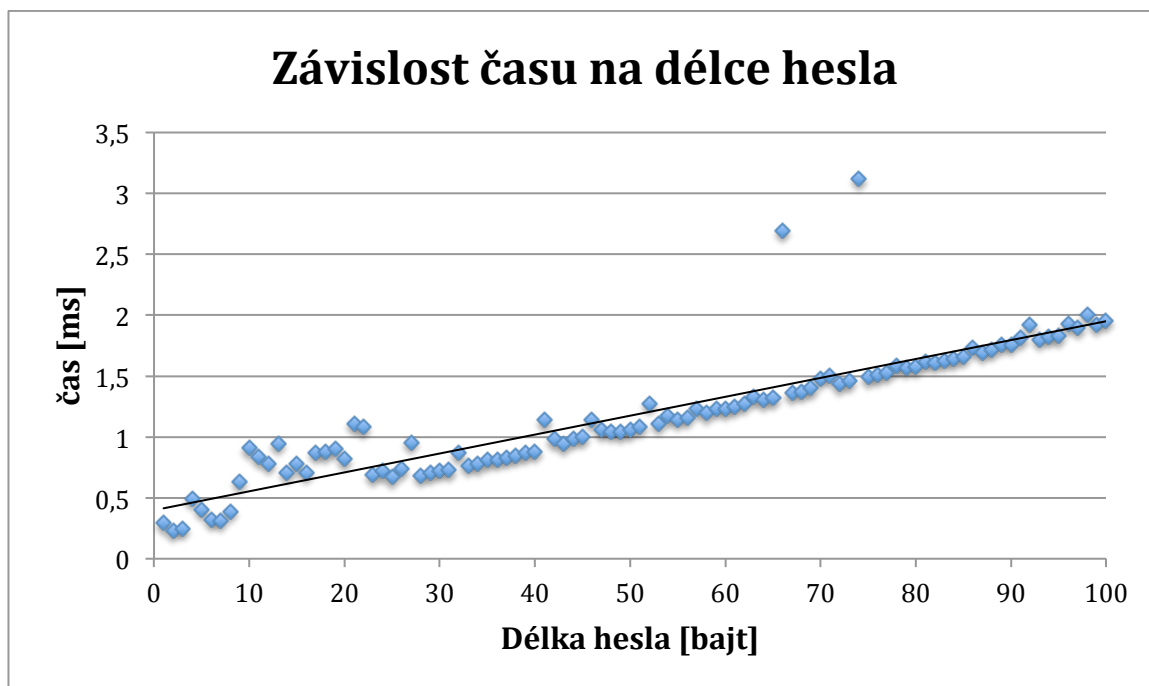
Operace byla podrobně popsána v kapitole 3.2. Výpočetní složitost a s tím spojená doba trvání operace pro vygenerování IV z uživatelského hesla je závislá hlavně na délce uživatelského hesla. Proto ji budeme značit $O(n)$, kdy n je délka uživatelské hesla v bajtech a je lineární k délce. Dále je nutno říci, že během operace dochází k dvojímu použití hashovací funkce SHA-256, pro kterou budeme používat výpočetní složitost $O(S)$ – konstantní. Výpočetní složitost musíme upravit z $O(n)$ na $O(p)$, protože při generování IV dochází k prodloužení uživatelského hesla v bajtech na délku $p = 11 \cdot n$ (p je délka hesla po prodloužení). Po první část, která je vmísení defaultního klíče do hesla, které probíhá cyk-

lem *for* o počtu kroků závislé na délce už prodloužené hesla, platí výpočetní složitost $O(p)$. Operaci generování mapy indexů můžeme shrnout s výpočetní složitostí $O(2 \cdot p) \rightarrow$ výpočetní složitost se změní na $O(3 \cdot p)$. Nakonec se provádí ještě přičtení uživatelské hesla po prodloužení a vmísení k hashi na základě mapy indexů, což má výpočetní složitost $O(p) \rightarrow$ z toho výsledná výpočetní složitost, včetně použití funkce SHA-256, bude $O(4 \cdot p + 2 \cdot O(S))$, kdy p je 11-násobkem uživatelského hesla. Pro zjištění doby potřebné na prodloužení hesla o velikost 1 bajt jsme provedli 100 pokusů s náhodným heslem velikosti 1 bajt. Průměrná doba pro prodloužení hesla o velikost 1 bajt nám vyšla 0,2ms a výsledek je patrný na následujícím grafu (odchyly způsobené vytížením a chodem programu pro výpočet průměrné doby zanedbáme).



Obrázek 63 - Graf času pro vygenerování IV z hesla o délce 1 bajt

Nyní otestujme lineární chování operace pro generování IV z hesla v závislosti na délce hesla n . Výsledek demonstruje následující graf.



Obrázek 64 - Graf závislosti času generování IV z hesla o délce n bajtů (znaků)

Z grafu si můžeme povšimnout, že doba trvání generování IV nevzrostla pro heslo o délce 100-násobně, ale jen 10-násobně. Tudíž provedme ještě jednu úpravu výpočetní složitosti na následující tvar $O((4 \cdot p + 2 \cdot O(S)) \cdot 0,1)$. Dobu trvání operace můžeme vyjádřit následujícím vztahem

$$t = k \cdot n \cdot 0,1 \text{ [ms]} \quad (91)$$

- t – celkový čas v milisekundách
- n – délka hesla v bajtech
- k - konstanta udávající průměrný čas potřebný vygenerování IV na 1 bajt (znak) hesla, závislá na hardwarové konfiguraci

V našem případě

$$t = 0,2 \cdot n \cdot 0,1 \text{ [ms]} \quad (92)$$

Na závěr je potřeba říct, že operace pro vygenerování se používá jen jednou, na začátku šifrování/dešifrování, proto se její doba moc nepromítne do celkové doby pro šifrování/dešifrování souboru, i kdyby bylo heslo o délce 100 bajtů (znaků).

4.5 Výkyvy doby trvání operací pro šifrování/dešifrování a režijních operací

V kapitolách 4.1, 4.2 a 4.4 si můžeme v grafech jednotlivých operací povšimnout velkých výkyvů v době trvání operací. Tyto nárůsty doby času se zdají být zapříčiněny z následujících důvodů:

1. Kolísání výkonnosti počítače v závislosti na aktuálním zatížení. (procesoru a paměti)
2. Režijní operace jazyka Python – údržba paměti (alokace, uvolnění, apod.)

Příčinou nejsou navržené operace.

4.6 Doba trvání šifrování/dešifrování souboru v závislosti na jeho velikosti

V této části budeme testovat vliv velikosti souboru na dobu šifrování/dešifrování. Celková doba by se měla odvíjet od počtu rund – r , počtu vstupních bloků – b , dílčích časů pro operace pro šifrování/dešifrování a operace pro režijní funkce a času pro ostatní režii (načítání bloků, ukládání, převody vstupů na řetězec, výstupů na bajty a další). Přitom čas pro ostatní režii se rozprostře v závislosti na počtu bloků a počtu rund. Čím větší bude vstupní soubor, tím menší čas pro režii připadne na jeden blok. Jinak bude doba šifrování/dešifrování téměř lineární k velikosti souboru. Přibližnou dobu trvání šifrování/dešifrování souboru o velikosti n bloků v r rundách, můžeme vyjádřit následovným vztahem.

$$t = t_s \cdot b \cdot r + t_p \cdot b \cdot r + t_m \cdot b \cdot r + t_{go} \cdot b \cdot r + t_{pk} \cdot b + t_{zst} \cdot b + t_{gIV} + T_{or} \quad (93)$$

- t – celkový čas šifrování/dešifrování souboru
- b – počet vstupních bloků k zašifrování/dešifrování
- r – počet rund
- t_s – čas potřebný pro substituci/inverzní operaci k substituci 1 bloku vstupních dat 1 rundou
- t_p – čas potřebný pro přičtení klíče/odečtení klíče na 1 bloku vstupních dat v 1 rundě
- t_m – čas potřebný pro mixování/inverzní operaci k mixování 1 bloku vstupních dat 1 rundou

- t_{go} – čas potřebný pro generování posloupnosti operací pro šifrování 1 bloku vstupních dat 1 rundou
- t_{pk} – čas potřebný pro posunutí klíče pro šifrování 1 bloku vstupních dat
- t_{zst} – čas potřebný pro zamíchání substituční tabulky pro šifrování 1 bloku vstupních dat
- t_{gIV} – čas potřebný pro vygenerování IV z hesla
- T_{or} – čas potřebný ostatní režii (ukládání bloků, načítání bloků, operace s řetězcí, obnova okna průběhu šifrování apod.)

Test závislosti doby šifrování/dešifrování na velikosti souboru byl proveden na dvou sadách souborů, za použití hesla - „test“. Test byl proveden pro každý soubor 1 až 3 rundami. Konfigurace hardwaru, na které proběhl test, je uvedena na začátku kapitoly 4.

1. Soubory o velikostech 1kB, 10kB, 100kB, 1MB a 10MB obsahující pouze znaky „a“.
2. Soubory o velikostech 1kB, 10kB, 100kB, 1MB a 10MB obsahující náhodné znaky.

Výsledné časy testů ukazuje následující tabulka (časy jsou závislé na aktuálním vytížení počítače).

Tabulka 4 - Časy pro šifrování a dešifrování souborů v závislosti na obsahu a velikosti

Šifrování				Dešifrování			
Soubory obsahující znaky 'a'				Soubory obsahující znaky 'a'			
Počet rund				Počet rund			
Čas[s]	1	2	3	Čas[s]	1	2	3
1kB	0,40	0,42	0,43	1kB	0,41	0,42	0,41
10kB	1,85	1,85	1,90	10kB	1,89	1,87	1,86
100kB	2,71	4,79	6,67	100kB	3,13	5,19	7,25
1MB	22,26	43,95	62,66	1MB	26,17	46,68	68,02
10MB	219,54	431,86	622,29	10MB	261,26	469,23	667,13
Soubory obsahující náhodné znaky				Soubory obsahující náhodné znaky			
Počet rund				Počet rund			
Čas[s]	1	2	3	Čas[s]	1	2	3
1kB	0,41	0,41	0,41	1kB	0,43	0,43	0,43
10kB	1,86	1,83	1,84	10kB	1,85	1,85	1,85
100kB	3,05	4,81	6,69	100kB	3,16	5,24	7,24
1MB	24,75	43,55	62,30	1MB	26,20	46,52	66,90
10MB	242,29	433,63	635,37	10MB	258,28	462,38	723,14

Nyní k závěrům, které jsou patrné z tabulky výše:

1. Do velikosti souboru 10kB má počet rund zanedbatelný vliv na čas pro šifrování/dešifrování. Je to díky tomu, že s počtem rund stoupá provedení operací k šifrování/dešifrování, ale nenarůstá počet vykonaných režijních operací a další režie. Tento čas pro operace začne převládat zhruba od velikosti 1MB, kdy se doba šifrování/dešifrování začíná výrazněji blížit lineární v závislosti na počtu rund.
2. Od velikosti 1MB se začíná projevovat linearita v závislosti na velikost souboru (počet bloků). Pro soubor 10MB je doba šifrování/dešifrování už přibližně desetkrát větší jako doba šifrování/dešifrování souboru o velikosti 1MB.
3. Opět důkaz asynchronnosti šifry \rightarrow která platí od velikosti souborů přibližně 1MB (projev vyšší výpočetní složitosti u inverzní operace k substituci)
4. A od velikosti souboru 100kB se ukazuje vliv obsahu šifrovaného/dešifrovaného souboru na době šifrování/dešifrování (projev výpočetní složitosti operací přičtení klíče/odečtení klíče).

4.7 Vliv hardwarové konfigurace na dobu šifrování/dešifrování souboru

Jak už bylo nastíněno výše, tak vliv hardwarové konfigurace na dobu šifrování/dešifrování je nesporný. Pro otestování vlivu hardwarové konfigurace na dobu šifrování/dešifrování byla navržena benchmarková funkce. Která otestuje dobu šifrování/dešifrování testovacího souboru o velikosti 100kB. Operace zjistí dobu, kterou při testování trvají operace pro šifrování/dešifrování, a dobu, kterou zabere vše ostatní. Zašifrování/dešifrování souboru se provádí 5 krát a časy se na konci zprůměrují. Označme si tyto časy jako t_1 a t_2 , kdy čas t_1 je čas pro aplikaci operací pro šifrování/dešifrování a t_2 je zbývajícím čas. To znamená že pro časy t_1 a t_2 platí následující vztah.

$$t_2 = t - t_1 \quad (94)$$

- t – celkový čas trvání funkce pro šifrování/dešifrování souboru

Nakonec na základě těchto dvou časů funkce aproximuje dobu trvání šifrování/dešifrování pro soubory o velikosti 1MB, 10MB a 100MB při použití 1 až 8 rund pro šifrování/dešifrování. Rovnice z které se aproximuje doba šifrování souborů o větší velikosti a o více rundách má následující tvar.

$$t = \left(\frac{v}{100000}\right) \cdot t_1 \cdot r + \left(\frac{v}{100000}\right) \cdot t_2 \quad (95)$$

- t - celkový čas trvání funkce pro šifrování/dešifrování souboru

- v – velikost souboru v bajtech, pro který chceme aproximovat dobu šifrování/dešifrování
- $t1$ – doba trvání operací pro šifrování/dešifrování
- $t2$ – doba trvání ostatní režie včetně režijních operací
- r – počet rund

Výpočet vychází z předpokladu lineárního chování šifry. Výpočet je aproximovaný, tedy ne naprosto přesný. Aproximace umožňuje výpočet doby i pro soubory a počet rund, pro které by otestování trvalo velkou dobu. Velikosti souborů 1MB, 10MB a 100MB, pro které jsou vypočítané aproximace doby trvání, byly také zvoleny záměrně. Důvody jsou popsány v předešlé kapitole. Funkce slouží zejména proto, aby si uživatel představil, jakou dobu zabere šifrování/dešifrování souboru v závislosti na velikosti souboru a počtu použitých rund při jeho hardwarové konfiguraci. Dále funkce aproximuje rychlost šifrování/dešifrování v kB/s pro danou velikost souboru a počet použitých rund. Výsledek funkce pro benchmark vypadá následovně.

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rundy	7 Rundy	8 Rundy
1. Soubor o velikosti 1MB ->	25s	45s	1min 5s	1min 25s	1min 45s	2min 4s	2min 24s	2min 44s
2. Soubor o velikosti 10MB ->	4min 12s	7min 31s	10min 49s	14min 8s	17min 26s	20min 45s	24min 3s	27min 22s
3. Soubor o velikosti 100MB ->	42min 0s	1hod 15min 6s	1hod 48min 11s	2hod 21min 17s	2hod 54min 23s	3hod 27min 28s	4hod 34s	4hod 33min 39s
Průměrná rychlost šifrování je	38.328 kB/s.	21.803 kB/s.	15.29 kB/s.	11.75 kB/s.	9.536 kB/s.	8.022 kB/s.	6.922 kB/s.	6.087 kB/s.
1. Soubor o velikosti 1MB ->	27s	49s	1min 10s	1min 32s	1min 53s	2min 15s	2min 37s	2min 58s
2. Soubor o velikosti 10MB ->	4min 30s	8min 7s	11min 43s	15min 19s	18min 55s	22min 31s	26min 7s	29min 43s
3. Soubor o velikosti 100MB ->	45min 4s	1hod 21min 6s	1hod 57min 7s	2hod 33min 8s	3hod 9min 10s	3hod 45min 11s	4hod 21min 12s	4hod 57min 14s
Průměrná rychlost dešifrování je	35.979 kB/s.	20.423 kB/s.	14.2 kB/s.	10.875 kB/s.	8.81 kB/s.	7.404 kB/s.	6.384 kB/s.	5.611 kB/s.

Rychlost je pouze orientační se může měnit v závislosti na různých faktorech, jako například:

- Výkon počítače
- Aktuální vytížení procesoru
- Rychlost šifrování/dešifrování – se může měnit, v závislosti na obsahu souboru.

Obrázek 65 - Výsledek funkce pro benchmark

Výsledek je sice aproximovaný, ale můžeme si povšimnout, že v porovnání s tabulkou 4 z předešlé kapitoly, poměrně přesný. Například doba šifrování souboru o velikosti 10MB ve třech rundách nám při testování vyšel na 622 a 635 sekund. Výsledek podle benchmarku je 649 sekund. Je tomu díky aproximování a částečně i díky obsahu souboru.

Jak bylo popsáno výše, implementace šifry a všech funkcí byla pomocí programovacího jazyk Python verze 3.x a na operačním systému Mac OS X. Abychom mohli otestovat vliv hardwarové konfigurace, tak z těchto důvodů bylo nutné celý program distribuovat na operační systém Microsoft Windows jako spustitelný soubor s příponou exe. Ne každý má operační systém Mac OS X a nainstalovaný Python verze 3.x se všemi potřebnými knihovnami pro kompilaci a spuštění. Distribuce byla provedena knihovnou pro Python verze

3.x *cx_freeze*, která hlavní soubor programu s příponou Main.py převede na spustitelný soubor s příponou exe a připojí všechny potřebné knihovny pro spuštění na operačním systému Microsoft Windows. S možností distribuce na operační systém Microsoft Windows bylo možné otestovat dobu šifrování/dešifrování na zařízeních s jinou hardwarovou konfigurací. Testování proběhlo na základě výpočtů navržené benchmarkové funkce a výsledky jsou vidět na následujících obrázcích. Jsou to obrázky vyřezané z okna pro zobrazení výsledků benchmarku.

Tabulka 5 – Hardwarové konfigurace pro výsledky benchmarků

Hardwarová konfigurace č.1	Intel Core2 Quad 2.5 GHz, 4 GB RAM DDR3, HDD WD 750 GB, Windows 8.1 Pro 64b
Hardwarová konfigurace č.2	Intel Core2Duo 2x 3,16 GHz, 4GB Ram, HDD WD 500GB, Windows 8.1 Pro 64bit
Hardwarová konfigurace č.3	AMD FX 6300(6x3,5GHz), 4GB RAM(DDR3 1600MHz), HDD 7200rpm, Windows 7
Hardwarová konfigurace č.4	Intel Core i7 3612QM, 4x2,1 GHz, 6GB DDR3, HDD Intel SSD 520 60GB, Windows 8.1 Pro 64-bit
Hardwarová konfigurace č.5	Intel Core2Duo E8400 2x3GHz, 4GB DDR2, 500GB HDD, Windows 7 Pro 64 Bit
Hardwarová konfigurace č.6	AMD Athlon X4 750K Quad 3.4Ghz, 8GB DDR3, 1TB WD Blue 7200 rpm, Win 8.1 Pro
Hardwarová konfigurace č.7	Intel Mobile Core 2 Duo 2.20GHz, 4,00GB Dual-Channel DDR2 328MHz, WD BLACK 500GB, 7 200 rpm, Windows 7 Professional 64-bit SP1
Hardwarová konfigurace č.8	Intel Core i5-3210M 2.5 GHz, 4 GB RAM, Windows 8 Pro 64-bit
Hardwarová konfigurace č.9	Intel Core i7-2670QM 2.2 GHz, 8 GB DDR3 1333 MHz), WD 750 GB 5400 RPM, Windows 8 Pro 64-bit
Hardwarová konfigurace č.10	CPU 2.53 GHz - 2 jádra, 4 GB RAM, TOSHIBA MK5055GSX 500 GB 5400 RPM, Windows 8 Pro 64-bit
Hardwarová konfigurace č.11	AMD Turion 64 X2 TL-60 2.00GHz - 2 jádra, 3 GB RAM, 160 GB HDD, Windows 8.1
Hardwarová konfigurace č.12	Intel Core i3-3225 2x3.30GHz, 8GB RAM, HDD 1 TB 7200rpm, Windows 8.1 Pro 64-bit
Hardwarová konfigurace č.13	Intel Core i5-3570k 3.40GHz(přetaktované na 4.2GHz) - 4 jádra, 8GB RAM, SSD, Windows 7 Home
Hardwarová konfigurace č.14	Intel Core i7 3517U 2.80GHz, 10 GB RAM DDR3, 232GB SSD + 24GB SSD, Windows 8.1 Pro 64-bit
Hardwarová konfigurace č.15	Intel Core i5-2410M 4x2,3GHz, 4GB DDR3-SDRAM, HDD WD 640GB, Win7Ultimate
Hardwarová konfigurace č.16	AMD E2-1800 APU, 4 GB RAM, HDD 200GB, Windows 7 Home Premium 64-bit
Hardwarová konfigurace č.17	Intel Core2duo P7570 2.27 GHz, 4GB, HDD 200GB, Windows 8 64-bit
Hardwarová konfigurace č.18	Intel Core i7 920 4x2.67 GHz, 16GB RAM DDR3 1600MHz, HDD 250GB 7200rpm, Windows 8.1 64-bit
Hardwarová konfigurace č.19	Intel Pentium 4 2.67 GHz, 1GB RAM, 8let staré HDD 320GB, Windows XP Pro SP3
Hardwarová konfigurace č.20	MacBook Air, Intel Core2duo i7 2GHz, 8GB RAM, APPLE SSD TS128E Media, Mac OS X 10.9.3

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	42s	1min 15s	1min 48s	2min 20s	2min 53s	3min 26s	3min 59s	4min 32s
2. Soubor o velikosti 10MB ->	6min 58s	12min 27s	17min 56s	23min 25s	28min 54s	34min 22s	39min 51s	45min 20s
3. Soubor o velikosti 100MB ->	11hod 9min 38s	2hod 4min 27s	2hod 59min 16s	3hod 54min 6s	4hod 48min 55s	5hod 43min 45s	6hod 38min 34s	7hod 33min 23s
Průměrná rychlost šifrování je	23.107 kB/s.	13.15 kB/s.	9.223 kB/s.	7.088 kB/s.	5.753 kB/s.	4.839 kB/s.	4.176 kB/s.	3.672 kB/s.
1. Soubor o velikosti 1MB ->	45s	1min 23s	2min 1s	2min 38s	3min 16s	3min 54s	4min 32s	5min 9s
2. Soubor o velikosti 10MB ->	7min 31s	13min 49s	20min 6s	26min 23s	32min 41s	38min 58s	45min 16s	51min 33s
3. Soubor o velikosti 100MB ->	11hod 15min 11s	2hod 18min 5s	3hod 21min 0s	4hod 23min 54s	5hod 26min 49s	6hod 29min 43s	7hod 32min 37s	8hod 35min 32s
Průměrná rychlost dešifrování je	21.92 kB/s.	12.042 kB/s.	8.288 kB/s.	6.316 kB/s.	5.102 kB/s.	4.279 kB/s.	3.685 kB/s.	3.235 kB/s.

Obrázek 66 - Výsledek benchmarku pro hardwarovou konfiguraci č. 1

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	35s	1min 31s	1min 31s	1min 59s	2min 26s	2min 54s	3min 22s	3min 50s
2. Soubor o velikosti 10MB ->	5min 53s	10min 31s	15min 9s	19min 46s	24min 24s	29min 2s	33min 39s	38min 17s
3. Soubor o velikosti 100MB ->	58min 52s	1hod 45min 9s	2hod 31min 25s	3hod 17min 42s	4hod 3min 59s	4hod 50min 16s	5hod 36min 32s	6hod 22min 49s
Průměrná rychlost šifrování je	27.367 kB/s.	15.588 kB/s.	10.936 kB/s.	8.406 kB/s.	6.823 kB/s.	5.74 kB/s.	4.953 kB/s.	4.356 kB/s.
1. Soubor o velikosti 1MB ->	37s	1min 8s	1min 39s	2min 10s	2min 41s	3min 12s	3min 43s	4min 14s
2. Soubor o velikosti 10MB ->	6min 8s	11min 18s	16min 28s	21min 38s	26min 48s	31min 57s	37min 7s	42min 17s
3. Soubor o velikosti 100MB ->	11hod 1min 22s	1hod 53min 0s	2hod 44min 39s	3hod 36min 18s	4hod 27min 56s	5hod 19min 35s	6hod 11min 13s	7hod 2min 52s
Průměrná rychlost dešifrování je	26.902 kB/s.	14.722 kB/s.	10.12 kB/s.	7.708 kB/s.	6.224 kB/s.	5.219 kB/s.	4.493 kB/s.	3.945 kB/s.

Obrázek 67 - Výsledek benchmarku pro hardwarovou konfiguraci č. 2

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	27s	49s	1min 10s	1min 31s	1min 53s	2min 14s	2min 35s	2min 57s
2. Soubor o velikosti 10MB ->	4min 33s	8min 7s	11min 40s	15min 14s	18min 47s	22min 21s	25min 55s	29min 28s
3. Soubor o velikosti 100MB ->	45min 33s	1hod 21min 9s	1hod 56min 44s	2hod 32min 19s	3hod 7min 55s	3hod 43min 30s	4hod 19min 5s	4hod 54min 41s
Průměrná rychlost šifrování je	35.184 kB/s.	20.141 kB/s.	14.156 kB/s.	10.89 kB/s.	8.843 kB/s.	7.442 kB/s.	6.424 kB/s.	5.65 kB/s.
1. Soubor o velikosti 1MB ->	30s	55s	1min 19s	1min 44s	2min 9s	2min 34s	2min 59s	3min 24s
2. Soubor o velikosti 10MB ->	4min 57s	9min 6s	13min 15s	17min 24s	21min 33s	25min 41s	29min 50s	33min 59s
3. Soubor o velikosti 100MB ->	49min 32s	1hod 31min 0s	2hod 12min 29s	2hod 53min 57s	3hod 35min 26s	4hod 16min 54s	4hod 58min 23s	5hod 39min 52s
Průměrná rychlost dešifrování je	33.279 kB/s.	18.274 kB/s.	12.575 kB/s.	9.583 kB/s.	7.74 kB/s.	6.491 kB/s.	5.59 kB/s.	4.908 kB/s.

Obrázek 68 - Výsledek benchmarku pro hardwarovou konfiguraci č. 3

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	26s	46s	1min 7s	1min 27s	1min 48s	2min 8s	2min 29s	2min 49s
2. Soubor o velikosti 10MB ->	4min 19s	7min 44s	11min 9s	14min 34s	17min 59s	21min 24s	24min 50s	28min 15s
3. Soubor o velikosti 100MB ->	43min 9s	1hod 17min 20s	1hod 51min 31s	2hod 25min 42s	2hod 59min 54s	3hod 34min 5s	4hod 8min 16s	4hod 42min 27s
Průměrná rychlost šifrování je	37.383 kB/s.	21.179 kB/s.	14.83 kB/s.	11.389 kB/s.	9.24 kB/s.	7.771 kB/s.	6.704 kB/s.	5.895 kB/s.
1. Soubor o velikosti 1MB ->	29s	53s	1min 17s	1min 41s	2min 5s	2min 29s	2min 53s	3min 17s
2. Soubor o velikosti 10MB ->	4min 49s	8min 50s	12min 50s	16min 50s	20min 50s	24min 50s	28min 50s	32min 51s
3. Soubor o velikosti 100MB ->	48min 14s	1hod 28min 16s	2hod 8min 18s	2hod 48min 20s	3hod 28min 21s	4hod 8min 23s	4hod 48min 25s	5hod 28min 26s
Průměrná rychlost dešifrování je	34.076 kB/s.	18.828 kB/s.	12.982 kB/s.	9.902 kB/s.	8.003 kB/s.	6.714 kB/s.	5.783 kB/s.	5.079 kB/s.

Obrázek 69 - Výsledek benchmarku pro hardwarovou konfiguraci č. 4

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	38s	1min 6s	1min 34s	2min 3s	2min 31s	2min 59s	3min 28s	3min 56s
2. Soubor o velikosti 10MB ->	6min 15s	10min 59s	15min 43s	20min 27s	25min 10s	29min 54s	34min 38s	39min 21s
3. Soubor o velikosti 100MB ->	1hod 2min 35s	1hod 49min 52s	2hod 37min 9s	3hod 24min 26s	4hod 11min 42s	4hod 58min 59s	5hod 46min 16s	6hod 33min 33s
Průměrná rychlost šifrování je	25.114 kB/s.	14.774 kB/s.	10.483 kB/s.	8.1 kB/s.	6.594 kB/s.	5.558 kB/s.	4.803 kB/s.	4.228 kB/s.
1. Soubor o velikosti 1MB ->	39s	1min 12s	1min 45s	2min 18s	2min 50s	3min 23s	3min 56s	4min 28s
2. Soubor o velikosti 10MB ->	6min 33s	12min 1s	17min 28s	22min 55s	28min 23s	33min 50s	39min 17s	44min 45s
3. Soubor o velikosti 100MB ->	1hod 5min 31s	2hod 5s	2hod 54min 39s	3hod 49min 13s	4hod 43min 47s	5hod 38min 20s	6hod 32min 54s	7hod 27min 28s
Průměrná rychlost dešifrování je	25.116 kB/s.	13.843 kB/s.	9.537 kB/s.	7.272 kB/s.	5.875 kB/s.	4.929 kB/s.	4.245 kB/s.	3.727 kB/s.

Obrázek 70 - Výsledek benchmarku pro hardwarovou konfiguraci č. 5

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	28s	50s	1min 12s	1min 34s	1min 56s	2min 18s	2min 40s	3min 3s
2. Soubor o velikosti 10MB ->	4min 37s	8min 18s	12min 0s	15min 41s	19min 22s	23min 3s	26min 45s	30min 26s
3. Soubor o velikosti 100MB ->	46min 13s	1hod 23min 5s	1hod 59min 57s	2hod 36min 50s	3hod 13min 42s	3hod 50min 34s	4hod 27min 27s	5hod 4min 19s
Průměrná rychlost šifrování je	35.026 kB/s.	19.739 kB/s.	13.796 kB/s.	10.585 kB/s.	8.583 kB/s.	7.216 kB/s.	6.224 kB/s.	5.472 kB/s.
1. Soubor o velikosti 1MB ->	32s	58s	1min 25s	1min 52s	2min 18s	2min 45s	3min 12s	3min 38s
2. Soubor o velikosti 10MB ->	5min 17s	9min 44s	14min 11s	18min 38s	23min 4s	27min 31s	31min 58s	36min 25s
3. Soubor o velikosti 100MB ->	52min 52s	1hod 37min 20s	2hod 21min 48s	3hod 6min 16s	3hod 50min 44s	4hod 35min 12s	5hod 19min 40s	6hod 4min 8s
Průměrná rychlost dešifrování je	31.223 kB/s.	17.095 kB/s.	11.753 kB/s.	8.952 kB/s.	7.229 kB/s.	6.062 kB/s.	5.219 kB/s.	4.582 kB/s.

Obrázek 71 - Výsledek benchmarku pro hardwarovou konfiguraci č. 6

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	46s	1min 21s	1min 56s	2min 31s	3min 7s	3min 42s	4min 17s	4min 52s
2. Soubor o velikosti 10MB ->	7min 40s	13min 32s	19min 23s	25min 15s	31min 6s	36min 57s	42min 49s	48min 40s
3. Soubor o velikosti 100MB ->	1hod 16min 41s	2hod 15min 16s	3hod 13min 50s	4hod 12min 25s	5hod 11min 0s	6hod 9min 35s	7hod 8min 10s	8hod 6min 44s
Průměrná rychlost šifrování je	20.635 kB/s.	12.027 kB/s.	8.507 kB/s.	6.563 kB/s.	5.339 kB/s.	4.498 kB/s.	3.885 kB/s.	3.419 kB/s.
1. Soubor o velikosti 1MB ->	49s	1min 31s	2min 13s	2min 54s	3min 36s	4min 17s	4min 59s	5min 41s
2. Soubor o velikosti 10MB ->	8min 15s	15min 11s	22min 7s	29min 2s	35min 58s	42min 54s	49min 50s	56min 46s
3. Soubor o velikosti 100MB ->	1hod 22min 27s	2hod 31min 46s	3hod 41min 5s	4hod 50min 25s	5hod 59min 44s	7hod 9min 3s	8hod 18min 22s	9hod 27min 42s
Průměrná rychlost dešifrování je	20.014 kB/s.	10.961 kB/s.	7.536 kB/s.	5.74 kB/s.	4.635 kB/s.	3.887 kB/s.	3.347 kB/s.	2.938 kB/s.

Obrázek 72 - Výsledek benchmarku pro hardwarovou konfiguraci č. 7

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	26s	46s	1min 6s	1min 27s	1min 47s	2min 7s	2min 27s	2min 48s
2. Soubor o velikosti 10MB ->	4min 17s	7min 40s	11min 2s	14min 25s	17min 48s	21min 11s	24min 34s	27min 57s
3. Soubor o velikosti 100MB ->	42min 47s	1hod 16min 35s	1hod 50min 24s	2hod 24min 12s	2hod 58min 0s	3hod 31min 49s	4hod 5min 37s	4hod 39min 25s
Průměrná rychlost šifrování je	37.662 kB/s.	21.378 kB/s.	14.98 kB/s.	11.508 kB/s.	9.337 kB/s.	7.854 kB/s.	6.776 kB/s.	5.959 kB/s.
1. Soubor o velikosti 1MB ->	28s	52s	1min 15s	1min 39s	2min 2s	2min 26s	2min 49s	3min 13s
2. Soubor o velikosti 10MB ->	4min 43s	8min 38s	12min 32s	16min 27s	20min 22s	24min 17s	28min 11s	32min 6s
3. Soubor o velikosti 100MB ->	47min 8s	1hod 26min 16s	2hod 5min 23s	2hod 44min 30s	3hod 23min 38s	4hod 2min 45s	4hod 41min 53s	5hod 21min 0s
Průměrná rychlost dešifrování je	34.877 kB/s.	19.267 kB/s.	13.283 kB/s.	10.132 kB/s.	8.188 kB/s.	6.87 kB/s.	5.917 kB/s.	5.196 kB/s.

Obrázek 73 - Výsledek benchmarku pro hardwarovou konfiguraci č. 8

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	[27s	[48s	[1min 9s	[1min 31s	[1min 52s	[2min 13s	[2min 34s	[2min 56s
2. Soubor o velikosti 10MB ->	[4min 28s	[8min 0s	[11min 33s	[15min 5s	[18min 38s	[22min 10s	[25min 43s	[29min 15s
3. Soubor o velikosti 100MB ->	[44min 35s	[1hod 20min 0s	[1hod 55min 25s	[2hod 30min 50s	[3hod 6min 15s	[3hod 41min 41s	[4hod 17min 6s	[4hod 52min 31s
Průměrná rychlost šifrování je	[36.107 kB/s.	[20.469 kB/s.	[14.336 kB/s.	[11.011 kB/s.	[8.933 kB/s.	[7.513 kB/s.	[6.482 kB/s.	[5.699 kB/s.
1. Soubor o velikosti 1MB ->	[30s	[55s	[1min 21s	[1min 46s	[2min 11s	[2min 36s	[3min 1s	[3min 26s
2. Soubor o velikosti 10MB ->	[5min 3s	[9min 14s	[13min 25s	[17min 36s	[21min 47s	[25min 58s	[30min 9s	[34min 20s
3. Soubor o velikosti 100MB ->	[50min 31s	[1hod 32min 21s	[2hod 14min 10s	[2hod 56min 0s	[3hod 37min 49s	[4hod 19min 39s	[5hod 1min 28s	[5hod 43min 18s
Průměrná rychlost dešifrování je	[32.442 kB/s.	[17.986 kB/s.	[12.414 kB/s.	[9.474 kB/s.	[7.659 kB/s.	[6.427 kB/s.	[5.537 kB/s.	[4.863 kB/s.

Obrázek 74 - Výsledek benchmarku pro hardwarovou konfiguraci č. 9

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	[42s	[1min 15s	[1min 48s	[2min 21s	[2min 53s	[3min 26s	[3min 59s	[4min 32s
2. Soubor o velikosti 10MB ->	[7min 0s	[12min 28s	[17min 57s	[23min 26s	[28min 55s	[34min 24s	[39min 53s	[45min 21s
3. Soubor o velikosti 100MB ->	[1hod 9min 55s	[2hod 4min 44s	[2hod 59min 32s	[3hod 54min 20s	[4hod 49min 9s	[5hod 43min 57s	[6hod 38min 46s	[7hod 33min 34s
Průměrná rychlost šifrování je	[22.957 kB/s.	[13.11 kB/s.	[9.206 kB/s.	[7.079 kB/s.	[5.747 kB/s.	[4.836 kB/s.	[4.174 kB/s.	[3.671 kB/s.
1. Soubor o velikosti 1MB ->	[46s	[1min 24s	[2min 2s	[2min 40s	[3min 18s	[3min 56s	[4min 34s	[5min 12s
2. Soubor o velikosti 10MB ->	[7min 37s	[13min 57s	[20min 16s	[26min 36s	[32min 56s	[39min 16s	[45min 36s	[51min 56s
3. Soubor o velikosti 100MB ->	[1hod 16min 8s	[2hod 19min 26s	[3hod 22min 45s	[4hod 26min 3s	[5hod 29min 21s	[6hod 32min 39s	[7hod 35min 57s	[8hod 39min 16s
Průměrná rychlost dešifrování je	[21.611 kB/s.	[11.924 kB/s.	[8.218 kB/s.	[6.267 kB/s.	[5.064 kB/s.	[4.249 kB/s.	[3.659 kB/s.	[3.213 kB/s.

Obrázek 75 - Výsledek benchmarku pro hardwarovou konfiguraci č. 10

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	[1min 7s	[2min 0s	[2min 53s	[3min 46s	[4min 39s	[5min 32s	[6min 25s	[7min 18s
2. Soubor o velikosti 10MB ->	[11min 11s	[20min 1s	[28min 51s	[37min 40s	[46min 30s	[55min 20s	[1hod 4min 10s	[1hod 12min 59s
3. Soubor o velikosti 100MB ->	[1hod 51min 50s	[3hod 20min 8s	[4hod 48min 25s	[6hod 16min 43s	[7hod 45min 1s	[9hod 13min 18s	[10hod 41min 36s	[12hod 9min 54s
Průměrná rychlost šifrování je	[14.476 kB/s.	[8.226 kB/s.	[5.766 kB/s.	[4.431 kB/s.	[3.595 kB/s.	[3.024 kB/s.	[2.61 kB/s.	[2.295 kB/s.
1. Soubor o velikosti 1MB ->	[1min 11s	[2min 10s	[3min 9s	[4min 8s	[5min 7s	[6min 6s	[7min 5s	[8min 5s
2. Soubor o velikosti 10MB ->	[11min 48s	[21min 39s	[31min 30s	[41min 21s	[51min 13s	[1hod 1min 4s	[1hod 10min 55s	[1hod 20min 46s
3. Soubor o velikosti 100MB ->	[1hod 57min 57s	[3hod 36min 29s	[5hod 15min 1s	[6hod 53min 33s	[8hod 32min 5s	[10hod 10min 37s	[11hod 49min 10s	[13hod 27min 42s
Průměrná rychlost dešifrování je	[13.967 kB/s.	[7.681 kB/s.	[5.288 kB/s.	[4.031 kB/s.	[3.256 kB/s.	[2.731 kB/s.	[2.352 kB/s.	[2.065 kB/s.

Obrázek 76 - Výsledek benchmarku pro hardwarovou konfiguraci č. 11

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	[24s	[42s	[1min 1s	[1min 20s	[1min 38s	[1min 57s	[2min 16s	[2min 34s
2. Soubor o velikosti 10MB ->	[3min 59s	[7min 5s	[10min 11s	[13min 17s	[16min 23s	[19min 29s	[22min 36s	[25min 42s
3. Soubor o velikosti 100MB ->	[39min 47s	[1hod 10min 49s	[1hod 41min 50s	[2hod 12min 52s	[2hod 43min 53s	[3hod 14min 54s	[3hod 45min 56s	[4hod 16min 57s
Průměrná rychlost šifrování je	[40.234 kB/s.	[23.067 kB/s.	[16.221 kB/s.	[12.482 kB/s.	[10.137 kB/s.	[8.532 kB/s.	[7.365 kB/s.	[6.478 kB/s.
1. Soubor o velikosti 1MB ->	[26s	[48s	[1min 9s	[1min 31s	[1min 53s	[2min 14s	[2min 36s	[2min 58s
2. Soubor o velikosti 10MB ->	[4min 19s	[7min 56s	[11min 33s	[15min 10s	[18min 47s	[22min 24s	[26min 1s	[29min 38s
3. Soubor o velikosti 100MB ->	[43min 13s	[1hod 19min 22s	[1hod 55min 31s	[2hod 31min 40s	[3hod 7min 49s	[3hod 43min 58s	[4hod 20min 7s	[4hod 56min 16s
Průměrná rychlost dešifrování je	[38.138 kB/s.	[20.954 kB/s.	[14.422 kB/s.	[10.991 kB/s.	[8.878 kB/s.	[7.446 kB/s.	[6.412 kB/s.	[5.63 kB/s.

Obrázek 77 - Výsledek benchmarku pro hardwarovou konfiguraci č. 12

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	[18s	[32s	[46s	[1min 0s	[1min 14s	[1min 28s	[1min 42s	[1min 56s
2. Soubor o velikosti 10MB ->	[3min 4s	[5min 23s	[7min 43s	[10min 2s	[12min 22s	[14min 41s	[17min 1s	[19min 20s
3. Soubor o velikosti 100MB ->	[30min 35s	[53min 50s	[1hod 17min 5s	[1hod 40min 20s	[2hod 3min 35s	[2hod 26min 50s	[2hod 50min 5s	[3hod 13min 20s
Průměrná rychlost šifrování je	[51.456 kB/s.	[30.176 kB/s.	[21.388 kB/s.	[16.517 kB/s.	[13.443 kB/s.	[11.329 kB/s.	[9.789 kB/s.	[8.616 kB/s.
1. Soubor o velikosti 1MB ->	[20s	[37s	[54s	[1min 11s	[1min 28s	[1min 45s	[2min 2s	[2min 19s
2. Soubor o velikosti 10MB ->	[3min 16s	[6min 7s	[8min 59s	[11min 50s	[14min 41s	[17min 32s	[20min 23s	[23min 14s
3. Soubor o velikosti 100MB ->	[32min 43s	[1hod 1min 14s	[1hod 29min 45s	[1hod 58min 16s	[2hod 26min 47s	[2hod 55min 18s	[3hod 23min 49s	[3hod 52min 20s
Průměrná rychlost dešifrování je	[50.83 kB/s.	[27.222 kB/s.	[18.582 kB/s.	[14.104 kB/s.	[11.365 kB/s.	[9.516 kB/s.	[8.185 kB/s.	[7.181 kB/s.

Obrázek 78 - Výsledek benchmarku pro hardwarovou konfiguraci č. 13

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	[26s	[47s	[1min 8s	[1min 29s	[1min 50s	[2min 11s	[2min 32s	[2min 53s
2. Soubor o velikosti 10MB ->	[4min 18s	[7min 49s	[11min 19s	[14min 50s	[18min 21s	[21min 51s	[25min 22s	[28min 53s
3. Soubor o velikosti 100MB ->	[43min 1s	[1hod 18min 8s	[1hod 53min 14s	[2hod 28min 20s	[3hod 3min 27s	[3hod 38min 33s	[4hod 13min 40s	[4hod 48min 46s
Průměrná rychlost šifrování je	[37.923 kB/s.	[21.055 kB/s.	[14.635 kB/s.	[11.201 kB/s.	[9.068 kB/s.	[7.617 kB/s.	[6.565 kB/s.	[5.769 kB/s.
1. Soubor o velikosti 1MB ->	[29s	[53s	[1min 17s	[1min 41s	[2min 4s	[2min 28s	[2min 52s	[3min 16s
2. Soubor o velikosti 10MB ->	[4min 49s	[8min 48s	[12min 47s	[16min 45s	[20min 44s	[24min 43s	[28min 42s	[32min 40s
3. Soubor o velikosti 100MB ->	[48min 11s	[1hod 27min 58s	[2hod 47min 46s	[3hod 47min 34s	[3hod 27min 21s	[4hod 7min 9s	[4hod 46min 57s	[5hod 26min 44s
Průměrná rychlost dešifrování je	[34.049 kB/s.	[18.881 kB/s.	[13.033 kB/s.	[9.947 kB/s.	[8.041 kB/s.	[6.748 kB/s.	[5.813 kB/s.	[5.106 kB/s.

Obrázek 79 - Výsledek benchmarku pro hardwarovou konfiguraci č. 14

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	32s	57s	1min 21s	1min 45s	2min 10s	2min 34s	2min 59s	3min 23s
2. Soubor o velikosti 10MB ->	5min 21s	9min 26s	13min 30s	17min 34s	21min 39s	25min 43s	29min 47s	33min 52s
3. Soubor o velikosti 100MB ->	53min 34s	1hod 34min 17s	2hod 15min 0s	2hod 55min 44s	3hod 36min 27s	4hod 17min 10s	4hod 57min 53s	5hod 38min 36s
Průměrná rychlost šifrování je	29.446 kB/s.	17.236 kB/s.	12.209 kB/s.	9.426 kB/s.	7.67 kB/s.	6.463 kB/s.	5.584 kB/s.	4.915 kB/s.
1. Soubor o velikosti 1MB ->	34s	1min 2s	1min 31s	1min 58s	2min 27s	2min 56s	3min 24s	3min 52s
2. Soubor o velikosti 10MB ->	5min 40s	10min 23s	15min 7s	19min 50s	24min 33s	29min 17s	34min 0s	38min 44s
3. Soubor o velikosti 100MB ->	56min 40s	1hod 43min 53s	2hod 31min 7s	3hod 18min 21s	4hod 5min 35s	4hod 52min 48s	5hod 40min 2s	6hod 27min 16s
Průměrná rychlost dešifrování je	29.053 kB/s.	16.003 kB/s.	11.023 kB/s.	8.404 kB/s.	6.79 kB/s.	5.696 kB/s.	4.905 kB/s.	4.307 kB/s.

Obrázek 80 - Výsledek benchmarku pro hardwarovou konfiguraci č. 15

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	1min 32s	2min 44s	3min 55s	5min 7s	6min 19s	7min 31s	8min 42s	9min 54s
2. Soubor o velikosti 10MB ->	15min 17s	27min 15s	39min 13s	51min 11s	1hod 3min 9s	1hod 15min 7s	1hod 27min 5s	1hod 39min 3s
3. Soubor o velikosti 100MB ->	2hod 32min 53s	4hod 32min 32s	6hod 32min 11s	8hod 31min 50s	10hod 31min 29s	12hod 31min 8s	14hod 30min 48s	16hod 30min 27s
Průměrná rychlost šifrování je	10.48 kB/s.	5.995 kB/s.	4.213 kB/s.	3.24 kB/s.	2.631 kB/s.	2.214 kB/s.	1.911 kB/s.	1.681 kB/s.
1. Soubor o velikosti 1MB ->	1min 44s	3min 10s	4min 35s	6min 1s	7min 27s	8min 53s	10min 19s	11min 45s
2. Soubor o velikosti 10MB ->	17min 17s	31min 35s	45min 54s	1hod 12s	1hod 14min 30s	1hod 28min 49s	1hod 43min 7s	1hod 57min 25s
3. Soubor o velikosti 100MB ->	2hod 52min 51s	5hod 15min 54s	7hod 38min 57s	10hod 2min 0s	12hod 25min 3s	14hod 48min 7s	17hod 11min 10s	19hod 34min 13s
Průměrná rychlost dešifrování je	9.511 kB/s.	5.264 kB/s.	3.631 kB/s.	2.771 kB/s.	2.239 kB/s.	1.879 kB/s.	1.619 kB/s.	1.422 kB/s.

Obrázek 81 - Výsledek benchmarku pro hardwarovou konfiguraci č. 16

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	45s	1min 20s	1min 56s	2min 31s	3min 6s	3min 41s	4min 17s	4min 52s
2. Soubor o velikosti 10MB ->	7min 30s	13min 23s	19min 16s	25min 8s	31min 1s	36min 53s	42min 46s	48min 39s
3. Soubor o velikosti 100MB ->	1hod 15min 3s	2hod 13min 49s	3hod 12min 36s	4hod 11min 22s	5hod 10min 8s	6hod 8min 55s	7hod 7min 41s	8hod 6min 28s
Průměrná rychlost šifrování je	21.381 kB/s.	12.218 kB/s.	8.582 kB/s.	6.6 kB/s.	5.359 kB/s.	4.509 kB/s.	3.892 kB/s.	3.423 kB/s.
1. Soubor o velikosti 1MB ->	52s	1min 35s	2min 18s	3min 1s	3min 44s	4min 27s	5min 11s	5min 54s
2. Soubor o velikosti 10MB ->	8min 38s	15min 49s	23min 1s	30min 12s	37min 23s	44min 35s	51min 46s	58min 58s
3. Soubor o velikosti 100MB ->	1hod 26min 18s	2hod 38min 12s	3hod 50min 6s	5hod 2min 0s	6hod 13min 54s	7hod 25min 48s	8hod 37min 42s	9hod 49min 36s
Průměrná rychlost dešifrování je	19.079 kB/s.	10.513 kB/s.	7.242 kB/s.	5.522 kB/s.	4.461 kB/s.	3.743 kB/s.	3.223 kB/s.	2.83 kB/s.

Obrázek 82 - Výsledek benchmarku pro hardwarovou konfiguraci č. 17

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	33s	57s	1min 21s	1min 46s	2min 10s	2min 34s	2min 59s	3min 23s
2. Soubor o velikosti 10MB ->	5min 28s	9min 31s	13min 34s	17min 37s	21min 39s	25min 42s	29min 45s	33min 48s
3. Soubor o velikosti 100MB ->	54min 41s	1hod 35min 9s	2hod 15min 38s	2hod 56min 6s	3hod 36min 34s	4hod 17min 2s	4hod 57min 30s	5hod 37min 58s
Průměrná rychlost šifrování je	28.207 kB/s.	16.958 kB/s.	12.121 kB/s.	9.397 kB/s.	7.664 kB/s.	6.468 kB/s.	5.594 kB/s.	4.928 kB/s.
1. Soubor o velikosti 1MB ->	36s	1min 6s	1min 36s	2min 6s	2min 36s	3min 6s	3min 36s	4min 6s
2. Soubor o velikosti 10MB ->	5min 58s	10min 58s	15min 59s	20min 59s	26min 0s	31min 1s	36min 1s	41min 2s
3. Soubor o velikosti 100MB ->	59min 35s	1hod 49min 41s	2hod 39min 48s	3hod 29min 54s	4hod 20min 0s	5hod 10min 6s	6hod 13s	6hod 50min 19s
Průměrná rychlost dešifrování je	27.68 kB/s.	15.165 kB/s.	10.428 kB/s.	7.944 kB/s.	6.415 kB/s.	5.38 kB/s.	4.632 kB/s.	4.066 kB/s.

Obrázek 83 - Výsledek benchmarku pro hardwarovou konfiguraci č. 18

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	1min 33s	2min 37s	3min 41s	4min 45s	5min 49s	6min 53s	7min 57s	9min 1s
2. Soubor o velikosti 10MB ->	15min 34s	26min 13s	36min 52s	47min 32s	58min 11s	1hod 8min 50s	1hod 19min 29s	1hod 30min 8s
3. Soubor o velikosti 100MB ->	2hod 35min 44s	4hod 22min 14s	6hod 8min 45s	7hod 55min 16s	9hod 41min 46s	11hod 28min 17s	13hod 14min 48s	15hod 1min 18s
Průměrná rychlost šifrování je	9.326 kB/s.	6.071 kB/s.	4.449 kB/s.	3.488 kB/s.	2.863 kB/s.	2.426 kB/s.	2.104 kB/s.	1.857 kB/s.
1. Soubor o velikosti 1MB ->	1min 26s	2min 38s	3min 50s	5min 1s	6min 13s	7min 24s	8min 36s	9min 47s
2. Soubor o velikosti 10MB ->	14min 25s	26min 20s	38min 16s	50min 11s	1hod 2min 6s	1hod 14min 2s	1hod 25min 57s	1hod 37min 52s
3. Soubor o velikosti 100MB ->	2hod 24min 10s	4hod 23min 23s	6hod 22min 36s	8hod 21min 49s	10hod 21min 2s	12hod 20min 16s	14hod 19min 29s	16hod 18min 42s
Průměrná rychlost dešifrování je	11.408 kB/s.	6.318 kB/s.	4.36 kB/s.	3.327 kB/s.	2.689 kB/s.	2.257 kB/s.	1.944 kB/s.	1.707 kB/s.

Obrázek 84 - Výsledek benchmarku pro hardwarovou konfiguraci č. 19

Počet rund	1 Runda	2 Rundy	3 Rundy	4 Rundy	5 Rundy	6 Rund	7 Rund	8 Rund
1. Soubor o velikosti 1MB ->	25s	45s	1min 5s	1min 25s	1min 45s	2min 4s	2min 24s	2min 44s
2. Soubor o velikosti 10MB ->	4min 12s	7min 31s	10min 49s	14min 8s	17min 26s	20min 45s	24min 3s	27min 22s
3. Soubor o velikosti 100MB ->	42min 0s	1hod 15min 6s	1hod 48min 11s	2hod 21min 17s	2hod 54min 23s	3hod 27min 28s	4hod 34s	4hod 33min 39s
Průměrná rychlost šifrování je	38.328 kB/s.	21.803 kB/s.	15.29 kB/s.	11.75 kB/s.	9.536 kB/s.	8.022 kB/s.	6.922 kB/s.	6.087 kB/s.
1. Soubor o velikosti 1MB ->	27s	49s	1min 10s	1min 32s	1min 53s	2min 15s	2min 37s	2min 58s
2. Soubor o velikosti 10MB ->	4min 30s	8min 7s	11min 43s	15min 19s	18min 55s	22min 31s	26min 7s	29min 43s
3. Soubor o velikosti 100MB ->	45min 4s	1hod 21min 6s	1hod 57min 7s	2hod 33min 8s	3hod 9min 10s	3hod 45min 11s	4hod 21min 12s	4hod 57min 14s
Průměrná rychlost dešifrování je	35.979 kB/s.	20.423 kB/s.	14.2 kB/s.	10.875 kB/s.	8.81 kB/s.	7.404 kB/s.	6.384 kB/s.	5.611 kB/s.

Obrázek 85 - Výsledek benchmarku pro hardwarovou konfiguraci č. 20

Bohužel se jedná jen o výsledky pouze na operačních systémech Microsoft Windows. V plánu bylo i otestování na vybraných mobilních zařízeních s operačním systémem Android, pro kterou doposud neexistuje nástroj, který by program v jazyce Python verze

3.x převedl na Android. Vyhlídkou do budoucna je vývojové prostředí QPython3, které funguje na Androidu. Nicméně převod Pythonu verze 3.x do verze pro QPython3 není zatím možný. A nyní k závěrům, které vyplývají z výše uvedených výsledků benchmarků pro různé hardwarové konfigurace.

1. Z důvodu implementace, která nevyužívá paralelizaci a více vláken, nehraje roli počet jader procesoru. Dochází k vytižení pouze maximálně jednoho jádra.
2. Největší roli, jak to podle výsledků vypadá, hrají diskové jednotky. Pokud program běžel na hardwarové konfiguraci, která měla diskovou jednotku SSD, byly výsledky lepší → doba šifrování/dešifrování se zkrátila. To vyplývá s faktu, že šifra je navržena tak, že pracuje pouze s 64 bajty dat, nikoliv celým souborem. Takže počet ukládání a čtení z disku je tolik, kolik je počtu bloků vstupního souboru. A jak je známo, v tomto ohledu jsou SSD disky bezkonkurenčně rychlejší, než pevné disky HDD. Takže disková jednotka ovlivňuje hlavně časy režie.
3. Co se týká procesorů, tak ty mají hlavně vliv na rychlost a dobu trvání jednotlivých operací (výpočet operací). Takže pokud má procesor lepší architekturu a vyšší frekvenci, také snižuje dobu šifrování/dešifrování souboru. Ale jak se ukazuje, není v tom takový rozdíl, jako v případě diskových jednotek. To by se ovšem změnilo, kdyby byla šifra implementovaná pro více jader.
4. Paměti počítače budou hrát taky svoji roli, protože jako je tomu u diskových jednotek, tak čím více vstupních bloků, tím více zápisů do paměti a čtení z paměti počítače.
5. Důležitou roli hraje i operační systém, zejména pokud se bude jednat o Mac OS X, který je navržený tak, aby využil maximální potenciál hardwarové konfigurace. Což ukazuje obrázek č. 85 s výsledky pro hardwarovou konfiguraci č 20. Konfigurace nemá procesor o frekvenci jako některé testované, ale i tak se jedná o jeden z nejlepších výsledků.
6. Nejzajímavější výsledek je pro hardwarovou konfiguraci č. 19, kdy se jedná o 8 let starý počítač s procesorem Intel Pentium 4 s frekvencí 2.67 GHz. Protože této výkonnosti už v dnešní době dosahují mobilní zařízení. Vyplývá to z tabulek benchmark testů pro procesory programem PassMark. Tyto tabulky poskytují výsledky jak pro běžné procesory, ale i pro procesory zařízení Android. Pokud si najdeme výše uvedený procesor v tabulkách, tak výsledek benchmarku programem PassMark pro tento procesor je 261 bodů.



Obrázek 86 - Výsledek benchmarku pro procesor Intel Pentium 4 2.66 GHz programem PassMark [8]

Pro mobilní zařízení s operačním systémem Android existuje program Android CPU Mark. Když srovnáme tabulku pro zařízení s Androidem s výsledkem pro procesor Intel Pentium 4 2.66 GHz – 261 bodů, tak ho to řadí v tabulce pro mobilní zařízení až na páté místo od konce, za HTC Wildfire S A510e. [9] Což znamená, že prakticky všechna mobilní zařízení by měla být srovnatelná, ne-li výkonnější jako hardwarová konfigurace č. 19. Tabulky jsou dostupné na adrese zdrojů [8] a [9]. Obrázek ukazující část tabulky výsledků benchmarku procesorů pro Android zařízení vypadá následovně



Obrázek 87 - Část tabulky benchmarku procesorů zařízení s Androidem [9]

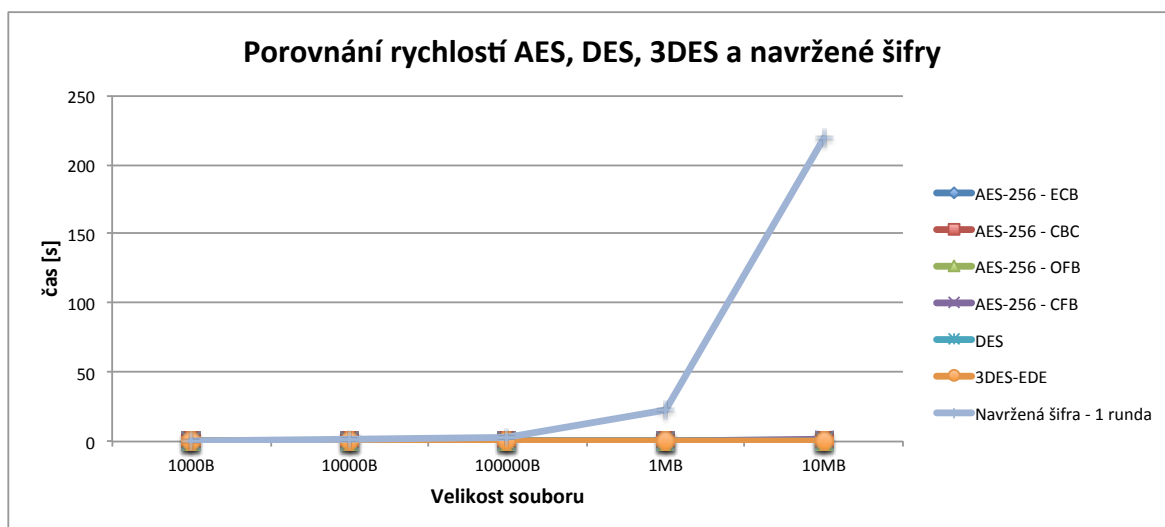
- Podle výsledků benchmarku na jiných hardwarových konfiguracích můžeme říct, že šifra dokáže šifrovat s rychlostí 31,5kB/s při jedné rundě na notebooku (počítačové stanici) a dle závěru číslo 6. s rychlostí přibližně >10kB/s při jedné rundě na jakémkoliv jiném mobilním zařízení.

4.8 Porovnání doby potřebné k šifrování navržené šifry s šiframi AES, DES a 3DES

Rychlost navržené šifry není příliš valná, ale to platí pro soubory větší 100kB. Vyzkoušejme jakou dobu trvání šifrování souborů budou mít šifry AES, DES a 3DES. Pro test použijeme stejné velikosti souborů, jako při testu navržené šifry – 1kB, 10kB, 100kB, 1MB a 10MB. U šifry AES budeme testovat i různé režimy činnosti – ECB, CBC, OFB a CFB. Výsledek testu ukazuje následující tabulka a graf

Tabulka 6 - Výsledné doby šifrování pro šifry AES, DES a 3DES v porovnání s navrženou šifrou

čas [s]	1000B	10000B	100000B	1MB	10MB
AES-256 - ECB	nelze	0,000089	0,000855	0,00808	0,093
AES-256 - CBC	nelze	0,000095	0,000922	0,00914	0,094
AES-256 - OFB	0,000029	0,000101	0,000931	0,00886	0,092
AES-256 - CFB	0,000137	0,00125	0,0123	0,132	1,31
DES	nelze	0,000217	0,00172	0,0172	0,171
3DES-EDE	nelze	0,000788	0,00469	0,0471	0,491
Navržená šifra - 1 runda	0,4	1,85	2,71	22,26	219,54



Obrázek 88 - Graf výsledků doby šifrování šifer AES, DES, 3DES a navržené šifry

Z výsledků je patrné, že navržená šifra začíná zaostávat od velikosti souboru 100kB. Do velikosti souboru 100kB je rozdíl nepatrný. Hodnoty v tabulce „nelze“ značí, že se nepodařilo otestovat rychlost šifrování souboru o velikost 1kB u daných šifer. Zdrojový kód knihovny použité pro šifrování PyCrypto verze 2.6.1 při pokusu o šifrování souboru o velikosti 1kB chybové hlášení. Každopádně by tyto časy situaci nijak nezměnily.

Musíme podotknout, že zdrojový kód knihovny PyCrypto, kterým jsou šifry implementované, je plně optimalizovaný a vyladěný na výkon. Což navržená šifra není. Díky optimalizaci by mohlo dojít k zkrácení doby šifrování navržené šifry.

4.9 Možnosti vylepšení rychlosti navržené šifry

Pokud bychom chtěli vylepšit rychlost navržené šifry, tedy zkrátit dobu šifrování a dešifrování souborů a nezměnit její bezpečnost. Můžeme udělat následující:

1. Zejména implementovat šifru tak, aby využívala více jader – paralelizovat ji, pokud je to možné.

2. Pracovat s daty v binárním tvaru, nepřevádět typ *bytearray* na řetězec, list, apod. – operace navíc. Každá operace navíc je samostatně nepatrný rozdíl v dobře šifrování, ale pokud se zvyšuje počet šifrovaných bajtů, počet operací roste s nimi.
3. Dále vylepšit její implementaci, použitím méně operací, rychlejších operací - optimalizovat implementaci šifry.
4. Implementovat ji v jiném programovacím jazyce, například C++, který je výpočetně rychlejší než programovací jazyk Python.
5. Provést důkladnou analýzu bezpečnosti navržené šifry a na jejím základě optimalizovat metody šifrování a případně je zredukovat. Ale jen v případě, kdy by se zmenšila bezpečnost šifry.

5 ANALÝZA BEZPEČNOSTI NAVRŽENÉ ŠIFRY

V této kapitole se budeme věnovat analýze bezpečnosti navržené šifry. Zhodnotíme slabá a silná místa, faktory které mohou vést k prolomení šifry a faktory které ztěžují prolomení šifry. Posoudíme možnost aplikování různých druhů kryptoanalýzy na navrženou šifru. Vypočítáme lavinový efekt v závislosti na různých faktorech a pro různé prvky šifry. Na závěr ji porovnáme s šiframi DES, 3DES a AES. Všechny závěry zhodnotíme a rozebereme.

5.1 Shrnutí vlastností šifry a co to znamená pro bezpečnost

Nejprve si shrneme, jaké vlastnosti má navržená šifra a co to znamená pro bezpečnost šifry.

5.1.1 Délka klíče

Šifra je navržená s délkou klíče 512 bitů, což je 64 znaků (bajtů). Pro bezpečnost šifry to znamená, že pokud bychom chtěli prolomit šifru hrubou silou (vyzkoušení všech možných klíčů), tak bychom museli otestovat 2^{512} klíčů, což je $1.341 \cdot 10^{154}$ možných klíčů. I kdybychom měli dostatečně velký výpočetní výkon k otestování 10^{10} klíčů za sekundu, trvalo by otestování všech klíčů $1.341 \cdot 10^{144}$ sekund. To je přibližně $3.1 \cdot 10^{126}$ krát víc než stáří vesmíru. Takže délka klíče poskytuje dostatečnou ochranu proti útoku hrubou silou. Při tom délka klíče 512 bitů nijak zásadně nesnižuje rychlost šifry, ale její vliv na bezpečnost je markantní.

5.1.2 Generování IV z uživatelského hesla pro šifrování

Jak už bylo popsáno výše, generování IV se provádí z hesla zadaného uživatelem a provádí se za použití hashovací funkce SHA-256. Na IV se opět aplikuje funkce SHA-256 a výsledný otisk se ukládá do souboru pro autorizaci uživatele. Použitím hashovací funkce SHA-256 docílíme toho, že je pro útočníka nemožné odvodit uživatelské heslo ani IV z hash otisku uloženého v souboru. Protože funkce SHA-256 je jednosměrná funkce. Celý proces generování IV z uživatelského hesla je navržený z toho důvodu, aby nebyla potřebné ukládat IV nebo jeho jinou distribuci. Uložení hashe IV není uložením IV , protože hashovací funkce SHA-256 odstranila jakýkoliv vztah hashe IV na IV . Během generování, kromě použití hashovací funkce SHA-256, dochází i k dalšímu zastření vztahu mezi ha-

shem *IV* a uživatelským heslem. Například vmísení znaků do hesla a zkombinování s hashem vmíseného hesla.

Generování *IV* z uživatelského hesla přináší však jednu slabou stránku, potenciální slabinu šifry, kterou by mohl útočník využít. Pokud by měl útočník k dispozici nástroj, který by převáděl uživatelská hesla na *IV*, tak by mohl na šifru aplikovat útok slovníkovou metodou nebo vyzkoušením všech kombinací hesel hrubou silou. To však za předpokladu, že uživatel použije pro šifrování slabé heslo. Útok slovníkovou metodou by znamenal razantní snížení počtu klíčů (*IV*), které mohly být použité k šifrování. Při nynější implementaci tu tato slabina je, protože zadávací pole pro heslo k šifrování zamezuje použití všech znaků z intervalu hodnot 0..255, ale jen znaky zadatelné na klávesnici. Ale nejvíce to záleží na samotném uživateli, jaké zvolí heslo. Na tento fakt je uživatel upozorněn v poznámkách v dolní části hlavního okna. Zde mu doporučujeme použít heslo o minimální délce 10 znaků a použití - malých/velkých písmen, číslic a speciálních znaků. Demonstrujme si tento fakt na příkladu. Uživatel si zvolí heslo a123456. Toto heslo je velice rychle prolomitelné slovníkovou metodou, ale i otestováním všech hesel délky 7 pro malá písmena a číslice. Protože pro otestování všech kombinací hesel obsahujících malá písmena a číslice o délce 7 nám stačí otestovat $36^7 = 78364164096$ hesel, což nezabere moc dlouhou dobu i při otestování milionu hesel za sekundu je to zhruba méně jak 22 hodin. Počet kombinací hesel v závislosti na použitých znacích a délce ukazuje následující tabulka.

Tabulka 7 - Počet kombinací hesel v závislosti na délce a použitých znacích

Znaková sada	heslo o délce 1	heslo o délce 2	heslo o délce 3	heslo o délce 4	heslo o délce 5	heslo o délce 6	heslo o délce 7	heslo o délce 8	heslo o délce 9	heslo o délce 10
[a..z]	26	676	17576	456976	11881376	308915776	8031810176	2,08827E+11	5,4295E+12	1,41167E+14
[A..Z]	26	676	17576	456976	11881376	308915776	8031810176	2,08827E+11	5,4295E+12	1,41167E+14
[0..9]	10	100	1000	10000	100000	1000000	10000000	100000000	1000000000	10000000000
[a..z],[A..Z]	52	2704	140608	7311616	380204032	19770609664	1,02807E+12	5,34597E+13	2,77991E+15	1,44555E+17
[a..z],[0..9]	36	1296	46656	1679616	60466176	2176782336	78364164096	2,82111E+12	1,0156E+14	3,65616E+15
[A..Z],[0..9]	36	1296	46656	1679616	60466176	2176782336	78364164096	2,82111E+12	1,0156E+14	3,65616E+15
[a..z],[A..Z],[0..9]	62	3844	238328	14776336	916132832	56800235584	3,52161E+12	2,1834E+14	1,35371E+16	8,39299E+17
[a..z],[A..Z],[0..9],[speciální znaky]	147	21609	3176523	466948881	68641485507	1,00903E+13	1,48327E+15	2,18041E+17	3,20521E+19	4,71165E+21

Speciální znaky - malá/velká písmena s interpunkcí, *@#\$%^&*(){}%"/[]";!~?,-.<>:\mezera - přibližně 85

Tabulka ukazuje, že použití hesla o délce 10 znaků, které bude obsahovat - malá/velká písmena, číslice a speciální znaky je dostatečné pro odolání útoku hrubou silou. Zároveň znemožňuje použití slovníkové metody. I kdybychom měli opět dostatečný výpočetní výkon k otestování 10^{10} hesel/sekundu. Tak k otestování hesla obsahující - malá/velká písmena, číslice a speciální znaky by nám trvalo $4,711 \cdot 10^{11}$ sekund, což je přibližně 14940 roků.

Další věc, která nahrává útočníkovi použít metodu hrubou silou pro otestování všech hesel je ta, že program vrací chybové hlášení uživateli pokud bylo zadáno špatné heslo a nespustí se dešifrování. Naprogramované to bylo tímto způsobem, aby uživatel nemusel čekat na dešifrování, aby se hned dozvěděl, že zadal špatné heslo. Proces dešifrování (zvláště u velkých souborů) trvá hodně dlouhou dobu. Pro uživatele by to bylo zdoluhavé. To však znamená, že ani útočník nemusí čekat na kompletní dešifrování souboru. Dešifrování se špatným heslem by bylo jinak samozřejmě možné, s tím, že dešifrovaný soubor by obsahoval jiný obsah než před zašifrováním.

Možnosti, jak řešit výše uvedená rizika jsou následující:

1. Uživatel sám zadá bezpečné heslo – klidně i delší, protože program není omezený délkou vstupního hesla. To však nebývá účinné, protože většina uživatelů si nevolí bezpečná hesla, protože si je nepamatují nebo se jim je nechce zadávat.
2. Před zašifrováním testovat sílu hesla, s tím, že uživateli nedovolíme šifrovat se slabým heslem. I to může být pro uživatele odrazující.
3. Další možností je nevracet chybové hlášení při zadání špatného hesla a spustit dešifrování. Což by při rychlosti šifry znemožnilo jakýkoliv útok hrubou silou i slovníkovou metodou na větší soubory. Což taky není nejlepší řešení, z důvodu komfortu pro uživatele.
4. Pravděpodobně nejlepší varianta je možnost použití souboru jako hesla pro šifrování. Uživatel by byl rád, že si heslo nemusí pamatovat a znemožnilo by to jakýkoliv útok hrubou silou nebo slovníkovou metodou na heslo. I když i v tomto je slabina. Když útočník nalezne soubor s heslem u uživatele → bude znát heslo na první pokus. Ale to už záleží na uživateli, jak bude soubor s heslem ukládat.

5.1.3 Použité operace pro šifrování

Operace byly podrobně popsány v kapitole 3, proto je zde popisovat už nebudeme. Ale budeme se věnovat tomu, co znamenají pro bezpečnost. Všechny operace jsou založené na operaci sčítání modulo. Proč je tato operace bezpečná (jednosměrná), že nejde ze zašifrovaných dat bez znalosti klíče odvodit vstupní data, to bylo popsáno a dokázáno v kapitole 1.2.1.2. Celkově se používají 3 různé operace k zašifrování dat. Substituce, několikanásobné přičítání klíče k datům a ještě operace mixování (transpozice). Operace by samy o sobě neměly až takový účinek, ale při jejich zkombinování se jejich síla zvýší tak, jako tomu je u šifer založených na vícenásobném použití S-Boxů a P-Boxů. Samostatná

monoalfabetické substituce je prolomitelná frekvenční analýzou. Polyalfabetická substituce indexem koincidence. Což sice neplatí pro datové soubory, ale jen pro textové soubory, protože neexistuje statistika, která by ukazovala frekvenci výskytu hodnot bajtů v souborech. Důležitá je také role použití klíče.

Bezpečnost se dále odvíjí od počtu použitých rund k šifrování. Čím více rund, tím je šifra bezpečnější. Pro vliv počtu rund navržené šifry by to chtělo podrobnou analýzu, ale můžeme říci, že navržená operace přičtení klíče k datům hraje největší roli. Je to z toho důvodu, že přičítání znaků klíče k datům dochází několikrát, ne pouze jednou. Přičtení, jak bylo rozebráno v kapitole 4.1.2, probíhá průměrně 2098 krát. Pokud bychom to rozpočítali na počet rund, kdy se jednorázově přičítá celý klíč (64 bajtů), tak zjistíme, že se samotná operace chová jako jednorázové přičtení klíče v 32 rundách. Tento fakt by mohl umožnit použití šifry i při jedné rundě. Nicméně stále platí, že čím více rund, tím větší bezpečnost. Na druhou stranu přímo úměrně roste doba šifrování. Možnost použití jedné rundy by se měla otestovat podrobněji. Budeme se tomu ještě věnovat v kapitole testování lavinového efektu a jiných. Naimplementovaná je možnost použití až 255 rund.

5.1.4 Generování posloupnosti operací pro šifrování a počet rund

Generování posloupností operací pro šifrování, si troufám tvrdit, že je nejvýznamnějším bezpečnostním prvkem navržené šifry. Samozřejmě to závisí i na počtu použitých rund. Generování posloupností operací zapřičiňuje skutečnost, že se navržená šifra nechová jako jedna šifra, ale v závislosti na klíči, jako více různých šifer. Což znemožňuje bližší analýzu šifrovaného textu, protože bez klíče znalosti nevíme, jaká byla použitá šifra. Tento fakt ztěžuje, ne-li vylučuje použití kryptoanalytických metod – lineární a diferenciální kryptoanalýza a zůstává pouze možnost prolomení šifry otestováním všech klíčů. Nebo navrženým nové speciální metody pro navrženou šifru. I tento fakt napomáhá použití pouze jedné rundy, protože při jedné rundě jsou data šifrované 1 ze 6 šifer, která je určena klíčem. Rozdíl je v použití jiné posloupnosti operací k zašifrování. Podrobně je to popsáno v kapitole 3.6. Ale pokud bychom použili 8 rund, bude se šifra chovat jako 6^8 jiných šifer, konkrétně 1679616 šifer. Pokud by se jednalo o 255 rund, počet šifer by vzrostl na $2,683 \cdot 10^{198}$. To je astronomické číslo, ale bohužel by se jednalo i o astronomickou dobu šifrování. Proto hraje důležitou roli možnost použití jen jedné rundy.

5.1.4.1 Otestování vlivu jiné posloupnosti operací na vstupní data

Pokud chceme brát předpoklad použití operací pro šifrování v jiném sledu za zvýšení bezpečnosti, je nutné otestovat změnu výstupu stejného vstupního bloku se stejným klíčem pro šifrování v závislosti právě na posloupnosti operací pro šifrování.

Otestujme tedy vstupní blok se samými znaky „a“ o velikosti 64 bajtů, zašifrovat stejným heslem pro šifrování – „test“, ale v jiném pořadí operací. Pro jednu rundu můžeme mít 6 posloupností, v jakém pořadí použít 3 operace.

1. substituce → přičtení klíče → mixování (označme si posloupnosti jako SPM)
2. substituce → mixování → přičtení klíče (označme si posloupnosti jako SMP)
3. mixování → přičtení klíče → substituce (označme si posloupnosti jako MPS)
4. mixování → substituce → přičtení klíče (označme si posloupnosti jako MSP)
5. přičtení klíče → substituce → mixování (označme si posloupnosti jako PSM)
6. přičtení klíče → mixování → substituce (označme si posloupnosti jako PMS)

Výsledek:

```
Vstup HEX - 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61
61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61 61

Výstup SPM HEX – ba b7 13 02 3c 86 d9 aa 9f 55 d1 2c 5a 02 80 11 ca 40 a5 ee 10 64 d8 67 6d f3 f6 dd 2f
60 00 6c 6d a1 24 35 6c cc 48 8a 6f 5c 39 09 d7 50 60 96 a7 df 68 44 56 b8 96 ed 2f c3 19 6f 34 14 f8 c5

Výstup SMP HEX – 0a 28 2c 91 16 36 bc 83 c8 bd 79 de cd 6e bd 7b ed 50 ff fd b5 e8 e3 97 66 a7 b1 04 4e
80 0a be 96 fe 2b 1a 6a c2 da 22 9c de d8 42 f3 76 51 56 a7 8d 43 44 f5 e6 72 dd 84 3c ca b8 53 78 3b c1

Výstup MPS HEX – 66 15 33 ca be 79 20 1c 0e f1 07 3c 1f f5 1b f9 8b bf 9b 7a 46 c7 95 a2 57 71 1f 58 9f
20 14 51 56 47 d9 ae 7b 12 f7 8b 36 ef 2a ea 40 bd 13 5e b9 47 5a 60 24 bb ce a0 41 93 aa ad 34 e2 86 bf

Výstup MSP HEX – cc df f3 09 50 35 ba f8 96 8a d9 11 b8 60 67 34 9f ed 02 c3 2f c5 68 13 5c ee 64 2c 6c
80 f6 60 aa d7 6d 02 86 39 6c 55 14 a5 d8 ca 19 3c 24 5a a7 56 6f 44 10 d1 b7 dd 40 2f 6f 96 6d 00 48 a1

Výstup PSM HEX – 20 ce a2 9b bd 7b 07 56 8b 8b bb 58 5e ae 20 3c ea 41 ef 71 24 1f 2a 1b d9 33 14 a0 93
f5 e2 f7 34 bf 13 79 9f 66 86 f1 aa 57 12 ca 47 be 51 ad b9 15 95 60 47 1f 0e bf 46 7a 40 5a f9 36 1c c7

Výstup PMS HEX – 18 c2 4b 21 b2 61 6e 92 87 fb 47 38 f5 74 5c d3 41 a0 48 21 68 ac 1d 8a c4 11 88 b5 20
f5 5e 93 1b bd be 8e aa 93 fe 82 d8 3c 12 e0 b4 ae 93 1d b9 ef ac 60 89 2e df bf 17 06 14 51 1b d1 88 ee
```

Tento výsledek napomáhá použití jen jedné rundy, protože bez znalosti klíče nelze určit, jaký má runda tvar (posloupnost operací). Dále napomáhá k vyloučení běžných známých i méně běžných kryptoanalytických metod, které jsou stavěny na pevnou strukturu šifry.

Nicméně neznámá to, že šifra nemá žádné slabiny. Každá šifra má určité slabiny. Pokud šifra nebude mít slabinu v proměnné struktuře, bude jinde.

5.1.5 Posunutí klíče – režim činnosti

Operace pro posunutí klíče (key-management, režim činnosti), jak už bylo popsáno výše, hraje taky velký vliv na bezpečnost. Znamená to použití odlišných klíčů na vstupní bloky dat → dva po sobě jdoucí bloky dat nejsou šifrované stejným klíčem. Posouvání dochází opět s variabilní strukturou určenou předchozím klíčem, vstupním a výstupním blokem dat. To opět znesnadňuje kryptoanalýzu, protože u šifer typu DES, 3DES a AES víme jaký režim činnosti byl použitý, tedy víme to s pravděpodobností 1 ku 4. U této šifry je počet režimů činnosti velké množství. Režim činnosti se určuje pro každý bajt klíče, tedy 64 krát. A každý bajt má vlastní režim činnosti v závislosti na součinu hodnot - bajtů klíče o jedna větší, vstupních dat o jedna větší a výstupních dat o jedna větší. To znamená, že režim činnosti pro jeden bajt klíče je určen číslem mezi 1 a 16777216. Což vypadá na první pohled krásně, ale po podrobnější analýze dojdeme pouze k 6 různým režimům činnosti. Pro důkaz lze použít vývojový diagram na obrázku č. 38, kdy číslo 6 odpovídá počtu možných cest tímto diagramem. Dalším důkazem ze tento zdrojový kód.

```
hodnoty = []
for i in range(256):
    for j in range(256):
        for k in range(256):
            soucin = (i + 1) * (j + 1) * (k + 1)
            hodnota = ''
            if soucin % 2 == 0:
                hodnota += 'a'
            if soucin % 3 == 0:
                hodnota += 'b'
            if soucin % 4 == 0:
                hodnota += 'c'
            else:
                hodnota += 'd'
        else:
            hodnota += 'e'
            if soucin % 3 == 0:
                hodnota += 'f'
            if soucin % 5 == 0:
                hodnota += 'g'
            else:
                hodnota += 'h'
        try:
            hodnoty.index(hodnota)
        except:
            hodnoty.append(hodnota)
print(len(hodnoty))

>>>
6
>>> hodnoty
['e', 'a', 'efh', 'abd', 'abc', 'efg']
```

Obrázek 89 - Zdrojový kód pro výpočet počtu režimů činnosti pro jeden bajt klíče a jeho výsledek

Program vypíše hodnotu 6, což odpovídá počtu možných cest navrženou strukturou větvení podmínek *if*. Nicméně to je pro jeden bajt klíče, pokud bychom chtěli vypočítat počet re-

žimů činnosti pro 64 bajtů klíče, dostaneme se na číslo $6^{64} = 6,344 \cdot 10^{49}$, což je podstatně větší číslo. Součin 3 bajtů umožňuje 16777216 různých výsledků, pokud tyto součiny rozložíme mezi varianty 1 až 6 z obrázku č. 89, tak rozložení bude následující.

Varianta č. 1 – ,e‘ – pro 614125 hodnot součinu.

Varianta č. 2 – ,a‘ – pro 4386086 hodnot součinu.

Varianta č. 3 – ,efh‘ – pro 746776 hodnot součinu.

Varianta č. 4 – ,abd‘ – pro 2213703 hodnot součinu.

Varianta č. 5 – ,abc‘ – pro 8080275 hodnot součinu.

Varianta č. 6 – ,efg‘ – pro 736251 hodnot součinu.

Ideální rozložení by bylo pro každou variantu stejné číslo, tedy $16777216/6 = 2796202$ hodnot součinu. Zlepšilo by to hodnotu lavinového efektu mezi dvěma po sobě jdoucími klíči. To opět napomáhá použití pouze jedné rundy. Tento fakt zapříčiňuje že po šifrování stejných vstupních bloků jsou si výstupní zašifrované bloky více rozdílné. Ukázka šifrování souborů obsahující stejné bloky bude v další části diplomové práce.

Výpočet nového bajtu klíče je na základě kombinací více vstupních bajtů (klíče, vstupního bloku a výstupního bloku). Výpočet je rovnice, která využívá různé operace a na konci dochází k aplikování operace modulo, takže zpětné není možné určit ze znalosti aktuálního klíče předešlý klíč. To znamená, že pokud by se útočníkovi podařilo odchytit nějaký z klíčů, dokáže dešifrovat pouze bloky za tímto klíčem, nikoliv předchozí bloky. Rovnice jsou zobrazeny opět na obrázku č. 31.

5.1.6 Zakódování bajtu udávající počet doplněných znaků (bajtů) necelého bloku

Operace byla popsána výše, proto ji tady nebudeme rozebírat dopodrobna. Na první pohled by se mohlo zdát, že se jedná o významnou slabinu, protože pokud bychom znali počet doplněných znaků, tak bychom z toho odvodili sumu klíče modulo 256. Ale tento fakt snižuje počet klíčů na 2^{504} . Dokážeme si to následovně, modulo si můžeme vyjádřit jako funkci, která vezme velké binární číslo a udělám s ním to, že v závislosti na hodnotě dělitele nechá pouze tolik posledních bitů, kolik má dělitel.

Příklad vypadá následovně:

Mějme klíč o délce 512 bitů v binární formě – 10010100...(496 nul a jedniček)...101100111. Pokud na tento klíč aplikujeme modulo 256 (binárně $2^9 = 100000000$),

tak nám zbyde poslední 8 bitů – 101100111, které nemají žádný vztah k předešlým 504 bitům. Proto je potřeba vypočítat předešlých 504 bitů hrubou silou. Což je stejné jako vyzkoušet 2^{504} klíčů. Pokud útočník nebude mít k dispozici znalost počtu doplněných znaků. Tak to opět navyšuje počet možných klíčů 63 krát, protože může dojít k doplnění 1..63 znaků a tato hodnota se přičítá k sumě z klíče.

5.1.7 Zakódování počtu rund

Zakódování počtu rund jsme podrobně popsali výše. Probíhá zakódováním pomocí *IV*. Pokud bychom brali možnost použití 0..255 rund, tak by se se zakódováním tohoto počtu neměnil počet možných klíčů. Jednalo by se o sumu 65 bajtů modulo 256, což nám nedává informaci o prvních 64 bajtech, což je 2^{512} . Nicméně, můžeme říct, že byla použita 1 runda s mnohem větší pravděpodobností, než 255 rund. Takže sice zašifrovaný soubor nenapovídá útočníkovi o tom, kolik bylo použito rund k šifrování, ale vyplatí se mu zkoušet dešifrovat soubor nejprve jednou rundou, dále dvěma, atd.

Pokud však nemá navrženou aplikaci, která umožňuje dešifrování, bez nutnosti zadávat počet rund pro dešifrování, ale počet rund se zjistí na základě *IV*, který je odvozen od uživatelského hesla pro šifrování. Tedy je lepší prolomit heslo. Vyřešit se to dá opět tak, že by uživatel musel zadat i počet rund pro dešifrování, což ale zase není pro uživatele tak pohodlné. Při použití dostatečně silného hesla se toho nemusíme obávat.

5.1.8 Absence rundovních klíčů

Šifra je navržena tak, že se klíč mění pouze tehdy, pokud se přistupuje k šifrování dalšího bloku dat. Nedochozí ke generování rundovních klíčů. Tento fakt by mohl oslabovat použití při více rundách. Na druhou stranu to dělá proces šifrování a dešifrování rychlejší. Ale jak bude ukážeme v další části – testování lavinového efektu, tak dochází ke změnám výstupu při použití 1 rundy a 2 rund při šifrování stejným heslem. Pokud bychom chtěli tento fakt změnit a posouvat klíč vždy po provedení rundy, tak by stačilo upravit dva řádky zdrojového kódu pro šifrování. Šlo by to udělat pomocí zaškrťovacího tlačítka v GUI, kdy by si mohl uživatel zvolit mezi bezpečnějším a normálním režimem. Absence rundovních klíčů, pokud to shrneme, znamená, že je pro všechny rundy použit stejný klíč. Nicméně generování posloupnosti operací zajišťuje, že je pro každou rundu zvolen jiný sled operací (Substituce, přičtení klíče a mixování), tedy 1 ze 6 posloupností. Takže můžeme říci, že generování posloupnosti v určité míře kompenzuje tento „nedostatek“.

5.1.9 Zamíchání substituční tabulky

Zamíchání substituční tabulky na základě klíče dochází při přechodu k šifrování dalšího bloku. Mohlo by k němu opět docházet vždy po každé rundě a to by zlepšilo bezpečnost šifry, ale zase by to prodloužilo dobu šifrování. Nezamíchání substituční tabulky po každé rundě znamená, že pro všechny substituce je použita stejná substituční tabulka, ale na jiná data, protože ty byly upravené ostatními operacemi. To znamená, že substituční tabulka se stejným klíčem, ale jinými vstupními daty, vrací jinou hodnotu zašifrovaných dat. Je to stejné, jako kdybychom šifrovali *ahoj* a *hola* pomocí klíče *zdar* za použití Vigenérový šifry. Popis Vigenérový šifry je dostupný ze zdrojů [1],[2] a [5]. Dostaneme jiný šifrový text.

Míchání substituční tabulky na základě klíče může teoreticky znamenat zamíchání substituční tabulky do takové podoby, že by její použití neznamenal změnu vstupního bloku po substituci. To může znamenat, že jeden blok dat, při jedné rundě, bude šifrován pouze metodami přičtení klíče a mixování. Ale i tak nevíme, v jakém pořadí. Dále nevíme, kdy k takové situaci došlo, takže ani nevíme, který blok byl „méně“ zašifrovaný, protože může dojít i k opačné situaci, že by výstupní blok po zašifrování měl vzhled srozumitelných dat (textu). Obě dvě varianty mohou nastat s velice malou pravděpodobností, protože se jedná o specifické hodnoty klíče, vstupních dat a substituční tabulky, s tím že počet substitučních tabulek je roven $256! = \text{cca } 8,578 \cdot 10^{506}$, což je nespočetně víc kombinací, jako je kombinací klíče.

5.1.10 Označení výstupního souboru příponou .encrypt

I tento fakt pomáhá útočníkovi v tom, že ví, který soubor je zašifrovaný. Pokud bychom k zašifrovanému souboru nepřipojovali příponu .encrypt, bylo by to určitým způsobem použití steganografie. Připojení přípony je zde opět z důvodu zpříjemnění používání programu pro uživatele, protože se před dešifrováním kontroluje soubor, jestli má příponu .encrypt. Pokud se zjistí, že nemá příponu .encrypt, dojde k chybovému hlášení a dešifrování se neprovádí.

5.1.11 Shrnutí a závěry

Z předešlých kapitol 5.1.1 až 5.1.10 vyplývá následující:

1. Největší hrozba spočívá v prolomení uživatelského hesla – mínus pro bezpečnost.
2. Šifra je silně závislá na klíči (uživatelském hesle) – plus pro bezpečnost (mínus, pokud je heslo slabé)
3. Prolomení klíče hrubou silou je výpočetně nemožné – plus pro bezpečnost.
4. Za předpokladu bezpečnosti hashovací funkce SHA-256 není snížena bezpečnost uložením hashe IV – plus pro bezpečnost.
5. Šifra se po prvotním analyzování jeví jako bezpečná i pro jednu rundu – nutná podrobnější analýza.
6. Bezpečnost šifry (klíče) není snížena na základě ukládání zakódovaných hodnot bajtů udávajících počet rund nebo počet doplněných znaků – délka klíče to vykompenzuje.
7. Šifra se chová jako více různých šifer, díky struktuře šifrování řízené klíčem – plus pro bezpečnost.

Možná řešení hrozeb a možná vylepšení šifry na základě analýzy:

1. Zejména přidání možnosti šifrování za využít obsahu souboru jako hesla pro šifrování. Generování IV to umožňuje (není závislé na délce vstupu). Nebo jiným způsobem zajistit použití bezpečného hesla pro šifrování.
2. Možnost vylepšení šifry pomocí rundovních klíčů.
3. Možnost zamíchání substituční tabulky po každé rundě.
4. Pro potvrzení bezpečnosti je potřebná zevrubnější analýza i pro určení minimálního počtu rund, aby byla šifra bezpečná pro použití.

5.2 Výpočet lavinového efektu pro navrženou šifru

V této části otestujeme šifru z hlediska lavinového efektu. Pokud má symetrická bloková šifra dobrý lavinový efekt, tak to znamená, že i při změně jednoho vstupního bitu, se změní alespoň polovina výstupních bitů. [5] U šifry AES se lavinového efektu docílilo vhodným návrhem substituční tabulky (S-Boxu), která má fixní charakter. [5] Navržená šifra nemá prakticky v žádném ohledu fixní charakter, až na aplikaci více rund, kdy se pro rundy negeneruje klíč a nemíchá se substituční tabulka. Proto lavinový efekt může měnit svojí hodnotu v závislosti na klíči (uživatelském hesle) a vstupních datech. Proto tato hod-

Obdobný výsledný lavinový efekt dostaneme při změně jednoho bitu na jakékoliv pozici. Lavinový efekt se bude zvyšovat v závislosti na počtu změněných bitů ve vstupu. Jestli se to týká i jiných bloků, provedme následující test:

šifrování 1 bloku (druhý v pořadí) – jedna runda a stejné vstupní heslo – „test“

1. změna jednoho bitu prvního bloku

Vstup 1 –

,aaa|aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa‘ (znak | odděluje bloky)

Výstup 1 HEX (druhý blok) – 92 53 21 67 31 db 5c fd 47 17 6b 40 6f 62 37 ad 40 0f ca 68 f2 88 7c 79 f4 95
15 82 c7 8c 36 41 a8 05 dd 36 6b ab 6b 49 5f d8 bb ed 36 d6 09 a6 ca 87 20 28 2e 1a 72 4a d5 15 5c f7 9b 83
80 a0

Vstup 2 –

,aaa**b**aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa‘ (znak | odděluje bloky)

Výstup 2 HEX (druhý blok) – 12 98 c1 72 53 bf 60 94 bf 5e 8e 99 78 fb cf f1 39 41 f1 f7 82 35 81 4b fd e0
e2 46 12 cb b5 dc e6 aa 25 38 8e 8d 55 1f 56 41 03 79 38 b5 7f d3 6d 24 6f 1b 5f 36 8d 62 24 e2 9f bc 69 b6
19 d1

Výsledný lavinový efekt – 52.93(271)

2. změna jednoho bitu druhého bloku

Vstup 1 –

,aa|aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaa**a**' (znak | odděluje bloky)

Výstup 1 HEX (druhý blok) – 92 53 21 67 31 db 5c fd 47 17 6b 40 6f 62 37 ad 40 0f ca 68 f2 88 7c 79 f4 95
15 82 c7 8c 36 41 a8 05 dd 36 6b ab 6b 49 5f d8 bb ed 36 d6 09 a6 ca 87 20 28 2e 1a 72 4a d5 15 5c f7 9b 83
80 a0

Vstup 2 –

,aa|aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaa**b**' (znak | odděluje bloky)

Výstup 2 HEX (druhý blok) – 92 53 21 67 31 db 5c fd 47 17 6b 40 6f 62 37 ad 40 0f ca 68 f2 88 7c 79 f4 95
15 82 c7 8c 36 41 a8 05 dd 36 6b ab 6b 49 5f d8 bb ed 36 d6 09 a6 ca 87 20 28 2e 1a 72 4a d5 15 5c f7 9b 83
80 a0

Výsledný lavinový efekt – 0.78(4)

Můžeme si všimnout:

1. Že změna jednoho bitu předchozího bloku, ovlivní výstup šifrování druhého (všech následujících) bloků, i když jsou na vstupu shodné. Zajištěno je to funkcí pro posunutí klíče, která na základě změněného vstupního předchozího bloku vygeneruje naprosto odlišný klíč.
2. Dále si můžeme všimnout, že „slabina“ z prvního testu se netýká jen prvního bloku. Tato „slabina“ se týká všech bloků, v kterých dochází k první změně. Pro příklad uvažujme tyto dva případy (znak „|“ odděluje dva bloky dat)
 - a. První vstupní data - a..a|a..a|a..a|a..a, druhé vstupní data – a..a|a..a|a..a|a..b → po zašifrování dojde v posledním bloku k velmi malým změnám a velmi nízké hodnotě lavinového efektu mezi posledními zašifrovanými bloky.
 - b. První vstupní data – a..b|a..a|a..a|a..a, druhé vstupní data – a..b|a..a|a..a|a..b → nyní po zašifrování dojde v posledním bloku k velkým změnám o hodnotách lavinového efektu mezi posledními zašifrovanými bloky. Na druhou stranu se první blok po zašifrování nebude moc lišit od bloku a..a po zašifrování.

Posunutí klíče pro následující blok závisí na předešlých stavech - vstupního bloku, výstupního bloku a klíče, proto se v případě 2b razantně změní výstup po zašifrování posledního bloku. Příklad 2a se děje ze dvou příčin. První je ta, že pro šifrování posledního bloku je použitý stejný klíč. Druhou je příčina ta, že operace pro zašifrování, pokud mají stejný klíč a data s malou změnou, vracejí zašifrovaný blok s malými změnami. Neboli šifrování operacemi je silně závislé na klíči, ale málo závislé na vstupních datech. Co toto zjištění znamená pro bezpečnost šifry.

- Týká se to zejména všech první bloků se změnou, které následující po stejných blocích. Všechny bloky, které se sice změnily, ale mají vstupní stejný klíč.
- Týká se to i prvního bloku, pokud je soubor o velikosti ≤ 64 bajtů, protože ten má vždycky stejný vstupní klíč (*IV* odvozený od vstupního hesla pro šifrování)
- Teoreticky se nejedná o slabinu šifru. Protože výstup se sice výrazně nemění při změně vstupu, ale taky výstup nenapovídá o hodnotě vstupu. Ať už operace pro šifrování používají stejný klíč nebo ne, tak se hodnota vstupních dat při šifrování změní zásadním způsobem, což jde

vidět z testu na frekvenční analýzu z kapitoly 5.3.1, kdy se znaky „a“ mění na jiné znaky. Nicméně, by bylo lepší tento fakt odstranit.

Teoretická řešení zjištěné „slabiny“ jsou následující:

1. Zajistit, aby klíč pro aktuálně šifrovaný blok byl upraven v závislosti na hodnotě aktuálně šifrovaných vstupních dat. Například, ke všech hodnotám bajtů aktuálního klíče přičíst sumu hodnot bajtů aktuálních dat. Nicméně, by to znamenalo, že si pro dešifrování musíme uchovat hodnotu této sumy. Abychom hodnotu sumy uchovali, tak hodnotu sumy následně zakódujeme klíčem pomocí funkce pro zakódování počtu rund (aktuálním klíčem – ne tím změněným) a uložíme před zašifrovaný výstupní blok dat. To by ovšem znamenalo nárůst velikosti zašifrovaného souboru o n bajtů dat, kdy n je počet vstupních bloků k šifrování. Takže soubor o velikosti 100000B (1563 bloků k zašifrování spolu s posledním doplněným blokem) by měl po zašifrování velikost 1001595B plus 64 bajtů hash IV , plus zakódovaný bajt udávající počet doplněných znaků a plus zakódovaný bajt udávající počet rund. Celkově to vychází na zvýšení velikosti přibližně o 1,6 %. Nárůst velikosti výstupního souboru není tak vysoký, ale není to nejlepší varianta. Navíc by každá suma musela být uložena pomocí aktuálního klíče, což by celkově znamenalo snížení velikosti klíče na 2^{504} , i když rezerva v délce klíče by toto oslabení klíče pokryla. Při dešifrování by se načetla hodnota bajtu udávající sumy hodnot vstupních bajtů dat pro změnu klíče. Hodnota by se dekodovala klíčem a přičetla stejně jako při šifrování, aby se dešifrovalo stejným klíčem, který byl použit pro šifrování. Pokud bychom hodnotu sumy neuchovávaly pro dešifrování, tak bychom sice docílili lavinového efektu pro všechny bloky, ale nebylo by možné dešifrování.
2. Druhou možností by byla změna dat před použitím šifrovaných operací tak, aby závislosti na místě se změnou došlo ke změně všech ostatních bajtů vstupních dat. Například, vytvořit funkci, která by postupně hodnotu všech bajtů vstupních dat přičetla ke zbývajícím hodnotám bajtů vstupních dat. Funkce by zajistila rozprostření změny jednoho bitu mezi ostatní bajty vstupních dat. Funkce by byla velmi výpočetně náročná, proto by se prováděla pro každý blok pouze jednou vždy před aplikací operací pro šifrování. Výpočetně náročná by byla z toho důvodu, že se musí pro každý bajt vstupních dat provést operace přičtení hodnoty bajtu 63 krát. Celkově by se provádělo $n \cdot (n - 1)$ operací, kdy n je velikost

bloku vstupních dat v bajtech (v našem případě je $n = 64$ bajtů) $\rightarrow 64 \cdot 63 = 4032$ operací přičtení hodnoty bajtu k jiné hodnotě bajtu. Operace by měla výpočetní složitost $O(b \cdot n \cdot (n - 1)) \rightarrow$ v našem případě $O(b \cdot 4032)$, kdy b je počet šifrovaných bloků dat. Stejný nárůst počtu potřebných operací by se týkal i dešifrování. Funkce by byla opět založena na operaci sčítání modulo. Zdrojový kód funkce by mohl vypadat následovně

```
def UpravaVstupu(vstupniBlok):
    vstupniBlokList = list(vstupniBlok)
    for i in range(delkaBloku):
        for j in range(delkaBloku):
            if j != i:
                vstupniBlokList[j] = chr((ord(vstupniBlokList[i]) + ord(vstupniBlokList[j])) % 256)
    vystupniBlok = ''.join(vstupniBlokList)
    return vystupniBlok
```

Obrázek 90 - Zdrojový kód funkce pro provázání změn vstupních dat

3. Že nedojde ke změně výstupu bitů předchozího bloku v závislosti na změně bitu následujícího bloku, vyplývá z návrhu šifru. Klíče se mění až po šifrování. Není možné, aby hodnota bajtu bloku na pozici i ovlivnila hodnoty bajtů bloků na pozicích menších jako i . Šifrování probíhá zleva doprava (ze shora dolů), nikoliv zprava doleva (z dolů nahoru).

5.2.2 Lavinový efekt jednoho vstupního bloku beze změny bitů - šifrování ve více rundách

V této části zkusíme otestovat nezměněný vstup na lavinový efekt při změně počtu použitých rund z jedné na dvě. Heslo pro šifrování bude opět stejné pro oba dva vstupy – „test“

Vstup 1 – „aa“

Výstup pro 1 rundu HEX – 66 15 33 ca be 79 20 1c 0e f1 07 3c 1f f5 1b f9 8b bf 9b 7a 46 c7 95 a2 57 71 1f 58 9f 20 14 51 56 47 d9 ae 7b 12 f7 8b 36 ef 2a ea 40 bd 13 5e b9 47 5a 60 24 bb ce a0 41 93 aa ad 34 e2 86 bf

Výstup pro 2 rundy HEX – 95 95 79 7f 72 2b b3 36 e7 52 60 fe b6 34 77 c3 26 6c 34 40 59 38 37 1c 03 5c 51 5d 17 14 14 41 e2 5a 3d 6b ce 55 79 09 1f 79 1e 6c 8e 0c c1 8c 9c d9 3b d7 93 d7 93 c9 a6 f5 d8 f3 11 c4 d5 1a

Výsledný lavinový efekt – 49.22(252)

Výsledný lavinový efekt se může měnit v závislosti na počtu rund, vstupních datech a vstupním klíči. Pokud se podíváme na výstup jako celé bajty, tak po zašifrování prvního

5.2.5 Lavinový efekt při generování *IV* ze zadaného hesla

V této části zkusíme otestovat funkci pro generování *IV* z hesla na lavinový efekt. To znamená, jestli když se změní vstupní heslo v jednom bitu, kolik se změní bitů ve vygenerovaném *IV*. Test provedeme pro hesla „aaaaaaa“ a „aaaaaab“. Už jenom použití hashovací funkce SHA-256 při generování *IV* z hesla by mělo zajistit lavinový efekt. Otestujme to.

Vstupní heslo 1 HEX – 61 61 61 61 61 61 61 61

Vstupní heslo 2 HEX – 61 61 61 61 61 61 61 62

IV hesla 1 HEX – 2e eb 3f f5 46 e3 d0 4b f7 c4 40 39 6a 2d 82 8f 7c 73 81 3f 83 10 18 58 ab 83 e8 9b 5a 00 2b 6f 73 4c 19 18 73 91 e1 42 51 39 97 9e e3 89 66 28 23 8b 1d 50 79 c3 bd 92 c9 e1 ac bc 4b 99 2c c7

IV hesla 2 HEX – 31 f7 3c 6d 1e e5 17 9f 8c e0 14 29 36 6a 9d b1 5a 65 21 1d 8b 19 51 ad c0 f0 91 f5 b5 b0 c0 7a c2 5b 99 25 be 47 f2 77 e4 a1 7a f6 f3 16 66 6d 27 b1 8d 7b 72 85 13 87 0f 49 59 ad b3 b5 9d 62

Výsledný lavinový efekt – 45.31(232)

Hodnota výsledného lavinového efektu se může měnit v závislosti na vstupním hesle.

5.2.6 Lavinový efekt při změně jednoho bitu vstupního hesla

V této části zkusíme otestovat lavinový efekt při zašifrování dvou stejných bloků s jiným vstupním heslem lišícím se v jednom bitu. Test provedeme pro hesla „aaaaaaa“ a „aaaaaab“.

Vstupní heslo 1 HEX – 61 61 61 61 61 61 61 61

Vstupní heslo 2 HEX – 61 61 61 61 61 61 61 62

Vstup – „aa“

Výstup pro heslo 1 HEX – e6 d7 05 58 d2 a0 84 31 90 5f 72 eb 9d 7f 48 d7 b2 95 e9 a2 d6 4b 66 00 8f d7 ed 8c 28 a4 46 4e 3f 21 84 f1 81 8a e8 50 1b eb 7f 4d a1 37 62 a6 b1 e6 6c 03 07 82 88 c4 1a bb 83 ab e4 b9 72 85

Výstup pro heslo 2 HEX – 7d bb e0 a9 95 16 4a 64 c5 77 06 7a a8 f6 57 84 41 b1 45 2e c9 6a 18 e8 38 2e 05 28 b9 2b 9a 2a a4 cf fa b6 da 4c 09 7e c0 83 9a 4a 16 6a c2 da 87 53 37 ee be 83 7a a5 b5 cb bd 2e d9 ab f1 3b

Výsledný lavinový efekt – 53.32(273)

Hodnota výsledného lavinového efektu se může měnit v závislosti na vstupním hesle.

de vypadat výstup. Například šifra AES je navržena tak, aby měla lavinový efekt o hodnotě přibližně 50 procent.

5.2.9 Zdrojový kód funkce pro výpočet lavinového efektu

Funkce pro výpočet využívá vstupní data převedené do podoby řetězce jejich binární hodnoty.

```
def LavinovyEfekt(vstup1Bin, vstup2Bin):  
    rozdíl = 0  
    for i in range(0, len(vstup1Bin)):  
        if (vstup1Bin[i] != vstup2Bin[i]):  
            rozdíl += 1  
    return (rozdíl / len(vstup1Bin), rozdíl)
```

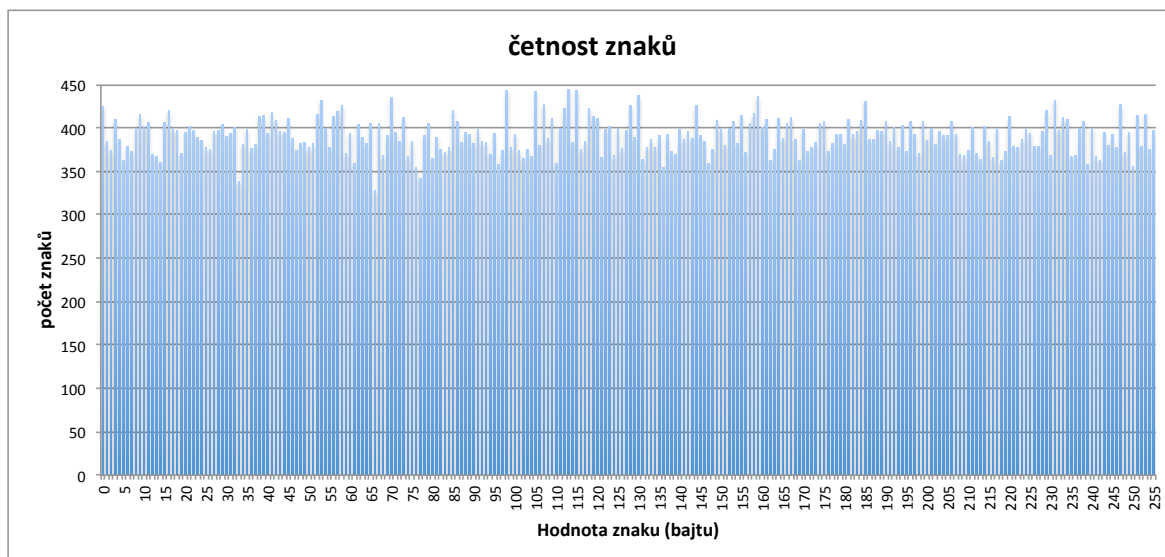
*Obrázek 91 - Zdrojový kód funkce
pro výpočet lavinového efektu*

5.3 Kryptoanalytické metody

V této kapitole uvedeme základní metody, které se používají ke kryptoanalýze šifer a zhodnotíme jejich využitelnost na navrženou šifru. Podrobnému popisu jednotlivých kryptoanalytických metod se věnovat nebudeme, to lze vyčíst z mnoha zdrojů, například [5].

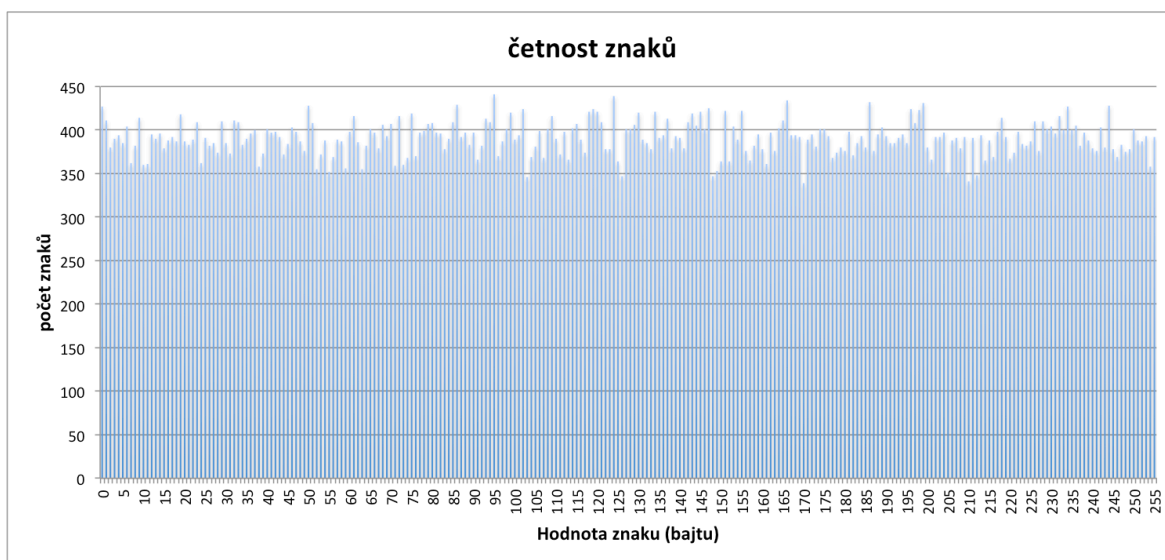
5.3.1 Frekvenční analýza

Frekvenční analýza se používá pro prolomení jednoduchých substitučních šifer. Využívá statistiky, která uvádí frekvenci výskytu jednotlivých znaků v určitém jazyce. Navržená šifra sice používá k šifrování substituci, ale řízenou klíčem, navíc používá další metody k šifrování, nejen substituci. Takže v našem případě frekvenční analýzu nelze použít, i kdyby tu ta možnost byla, neexistuje statistika, která by udávala frekvenci výskytu bajtů v souborech. Frekvenci výskytu znaků pro český (jakýkoliv) jazyk nemůžeme použít z prostého důvodu. Výstupem šifrování nejsou pouze znaky abecedy, ale hodnoty z intervalu 0..255 představující hodnotu znaku v ASCII tabulce. I když by to mohlo být zajímavé. Z toho důvodu nelze použít ani frekvenční analýzu na Caesarovu šifru použitou na soubor dat. Pro zajímavost můžeme zkusit vypočítat četnost znaků (bajtů) s hodnotami 0..255 pro zašifrovaný soubor o velikost 100kB, který před zašifrováním obsahoval samé znaky „a“ – hodnota 97 a byl zašifrovaný pomocí navržené šifry.



Obrázek 92 - Četnost znaků zašifrovaného souboru na základě frekvenční analýzy - navržená šifra

Pro porovnání následuje četnost znaků po zašifrování stejného vstupního souboru obsahujícího samá písmena „a“ šifrou AES-256.



Obrázek 93 - Četnost znaků zašifrovaného souboru na základě frekvenční analýzy – AES-256

Původně před zašifrováním byla četnost výskytů pro všechny znaky kromě „a“ rovna 0 a znaku „a“ rovna 100000. Nyní vidíme, že se četnost výskytu pěkně rozprostřela na všechny znaky. Další důkaz kvalitního lavinového efektu. Frekvenční analýza by byla použitelná za předpokladu, že by z ní vyšel graf četností, kdy by měl jiný znak četnost 100000.

5.3.1.1 Zdrojový kód funkce pro výpočet četnosti znaků výše uvedené frekvenční analýzy.

```
def FrekvencniAnalyza(zasifrovanyText):  
    tabulkaCetnosti = []  
    for i in range(256):  
        tabulkaCetnosti.append(0)  
    for i in range(0, len(zasifrovanyText)):  
        tabulkaCetnosti[zasifrovanyText[i]] += 1  
    return tabulkaCetnosti
```

Obrázek 94 - Zdrojový kód funkce pro výpočet četnosti znaků zašifrovaného textu

5.3.2 Index koincidence

Co se týká indexu koincidence, teak opět metoda týká monoalfabetických a polyalfabetických substitučních šifer. Nicméně pro jejich fungování je nutné znát pravděpodobnost výskytu znaků použité abecedy. V našem případě hodnot bajtů. Tato tabulka pravděpodobnosti doposud neexistuje, takže tato kryptoanalytická metoda tady nepřipadá v úvahu.

5.3.3 Lineární a diferenciální kryptoanalýza

Určit zda je navržená šifra odolná vůči výše uvedeným dvěma kryptoanalytickým metodám je obtížnější. Protože jak je uvedeno v literatuře [10] věnované moderním metodám kryptoanalýzy, tak se tyto metody používají na blokové šifry. Nicméně se jejich aplikace odvíjí od struktury šifry a v knize jsou tyto dvě metody použité pouze na blokové šifry pevné struktury, které využívají pouze operace XOR. V literatuře [10] je sice zmíněno, že diferenciální kryptoanalýza se dá použít na různé typy šifer, ale nikde jsem nenašel zdroj, který by uváděl využití těchto metod na kryptoanalýzu například šifry IDEA, která je použitými operacemi podobná navržené šifře.

Minimálně, pokud by se tyto metody daly aplikovat na navrženou šifru, jsou podmínky proměnlivou strukturou velice ztíženy, protože aby byl útočník aplikovat tyto kryptoanalytické metody, musí znát strukturu šifry, což u navržené šifry bez znalosti klíče pro šifrování není možné. A s počtem použitých rund je tento úkol stanovit strukturu ještě obtížnější.

Další důvod, který napomáhá faktu, že by tyto metody neměly vliv na navrženou šifru, je hodnota lavinového efektu. Za předpokladu, pokud nedojde k výše zmíněné „slabiny“ v kapitole 5.2.1. Pro zlepšení odolnosti by bylo vhodnější aplikovat jedno z řešení

pro „slabinu“ z kapitoly 5.2.1. V ostatních případech, se hodnota lavinového efektu pohybuje okolo hodnoty 50, což je ideální hodnota pro zabránění použití lineární a diferenciální kryptoanalýzy. Tyto dvě metody jsou totiž neúčinnější na šifry s lavinovým efektem blížícím se hodnotě 100 nebo 0. Například AES je navržen s lavinovým efektem okolo hodnoty 50, díky návrhu Rijndael S-Boxu. [10]

5.3.4 Jiné pokročilé kryptoanalytické metody

Například metoda Narrow-Bicliques, která se zdá být použitelná i na šifru AES i na šifru IDEA. [11] Může být pro studium zajímavá a do budoucna může najít svoje využití. Metoda je na snížení počtu testovaných klíčů ze struktury šifry [11]. Další metody kryptoanalýzy jsou Boomerang, meet-in-the-middle, Related-key method[10]. Ale tyto metody nejsou až tak běžné na rozdíl od lineární a diferenciální kryptoanalýzy.

5.3.5 Útok hrubou silou

Metoda, která jde použít na všechny šifry, teda kromě Vernamovy šifry, která je nedešifrovatelná i za využití otestováním všech kombinací klíčů. Nicméně, útok hrubou silou je na navrženou šifru možný, ale velmi jej ztěžuje fakt, že klíč má délku 512 bitů, protože při dnešním výpočetním výkonu není možné otestovat všechny klíče. Ani za použití kvantových počítačů. [5]

Důležité je říct, že na navrženou šifru je možné provést útok slovníkovou metodou a nebo útok na prolomení hesla pro šifrování. Nemusí se jednat pouze o útok na klíč. Při stávající implementaci, při použití souboru jako vstupního hesla to možné už nebude.

5.4 Posouzení bezpečnosti s šiframi DES, 3DES a AES

V této části posoudíme bezpečnost navržené šifry v porovnání s šiframi DES, 3DES a AES.

5.4.1 Navržená šifra versus DES

Nejprve k porovnání navržené šifry se šifrou DES. Šifra DES se v dnešní době už nepoužívá, je zastaralá a prolomitelná. Šifra má velice krátký klíč. Co se týká klíče, má navržená šifra mnohem větší bezpečnost (pokud zajistíme spolehlivé vstupní heslo). Další aspekty, jako proměnná struktura rundy, proměnný režim činnosti, proměnná substituční tabulka činí navrženou šifru mnohem bezpečnější jako je tomu u DESu.

5.4.2 Navržená šifra versus 3DES

Hodnocení je podobné jako v předchozí kapitole. 3DES je totiž založený na DESu, s tím, že má trojnásobnou délku klíče (ve většině jeho mutací). Z hlediska délky klíče má opět navrch navržená šifra. Ale není vyloučeno, že neexistuje slabina navržené šifry, ale podle dosavadní analýzy si troufám tvrdit, že má navržená šifra výhody. Opět pokud zmíním proměnnou strukturu rund, režimu činnosti a substituční tabulky. Jako výhodu bych uvedl i použití operace sčítání modulo namísto operace XOR, tím se podobá šifře IDEA, pro kterou bylo doposud nalezeno pouze teoretické prolomení metodou Narrow-Bicliques, popsané v literatuře [11], která popisuje, že je možné snížit dobu útoku hrubou silou až 2^{18} krát. Ale to jsou vše šifry s pevnou strukturou, pro kterou jsou tyto metody navrženy.

5.4.3 Navržená šifra versus AES

Nejhůře se hodnotí navržená šifra v porovnání s šifrou AES. Šifra AES je považována za jednu z nejbezpečnějších blokových šifer dnešní doby. Ačkoliv metoda Narrow-Bicliques počítá s možností oslabení i šifry AES. Ovšem jen z hlediska útoku hrubou silou, kdy metoda redukuje počet potřebných klíčů k otestování. Pokud bychom navrženou šifru i šifru AES chtěli prolomit otestováním klíčů na základě redukce klíčů, tak navržená šifra má větší počet klíčů potřebných k otestování. I když při dnešním běžném výpočetním výkonu není možné otestovat všechny klíče v rozumném čase.

AES je postavený tak, aby odolal všem doposud známým metodám používaných pro kryptoanalýzu. Zejména aby odolal lineární a diferenciální kryptoanalýze. Navržená šifra se zdá být taky odolnou proti těmto metodám, ať kvůli její celé proměnlivosti, tak na základě použité operace sčítání modulo. Posouzení jestli je bezpečnější pro jednu rundu, by bylo zajímavé. Nebo stanovení minimálního počtu potřebných rund k zajištění dostatečné bezpečnosti.

Minimálně proměnlivá struktura rundy a režimu činnosti je zajímavý prvek, kterým se AES nemůže chlubit, protože nestačí prolomit jednu strukturu. Bez znalosti klíče neznáme strukturu šifry, u ostatních šifer, i šifry AES, ano.

Výhody v porovnání se šifrou AES:

- délka klíče
- proměnlivá struktura – režimu činnosti, struktury rundy

Nevýhody v porovnání se šifrou AES:

- operace jsou závislé jen na klíči, nikoliv na vstupních datech aktuálně šifrovaného bloku → AES (řezim činnosti CBC a EBC) má dobrý lavinový efekt i pro bloky se změnou v jednom bitu za použití stejného klíče.

Šifra se chová srovnatelně podobně při rozložení hodnot znaků na výstupu. Viz. výsledky frekvenční analýzy v kapitole 5.3.1. U šifry AES s režimem činnosti CFB a OFB se chová AES jako navržená šifra v ohledu na „slabinu“ zjištěnou v kapitole 5.2.1. Tento fakt se dá vyzkoušet na webových stránkách [12], kde je možné si vyzkoušet fungování šifry AES v různých režimech činnostech a zobrazit výsledek.

5.5 Zhodnocení závěrů z analýzy bezpečnosti

1. Z testu na lavinový efekt bylo zjištěno, že je operace jsou vysoce závislé na klíči (při změně klíče je lavinový efekt zajištěn), ale slabě závislé na vstupních datech pro šifrování, pokud se nemění klíč. Teoretická řešení jsou uvedena v kapitole 5.2.1. Ačkoliv se nemusí jednat o veliké oslabení šifry, protože se podobně chová i AES v režimech činnosti CFB a OFB. Vyplyývá to z testů šifry AES na webových stránkách [12].
2. Mnoho faktů, jako lavinový efekt, proměnný režim činnosti, proměnná struktura rundy, několikanásobné přičtení bajtů klíče k datům na každou rundu, nasvědčují bezpečnému použití šifry v jedné rundě.

3. Největší slabinou, která může při aktuální implementaci vzniknout a která byla popsána při rozebírání generování IV z hesla, je právě heslo. Pokud nebude bezpečné heslo je mnohem snadnější prolomit hrubou silou heslo, než IV . Dalším faktorem je chybové hlášení při zadání špatného hesla k dešifrování, jenž umožňuje zrychlení otestování všech hesel. Možné řešení, jako šifrování souborem nebo vypnutí chybového hlášení, byly popsány výše.
4. Operace, která je nejvíce využívána v operacích pro šifrování a dešifrování, je sčítání modulo. Operace se jeví jako bezpečná a jednosměrná. I podle literatury [10] by to nasvědčovalo tomu, že nelze použít metody - diferenciální a lineární kryptoanalýza, které jsou v uvedené literatuře použity pouze na šifry využívající operace XOR.
5. Běžně používané metody pro kryptoanalýzu vyžadují znalost struktury šifry. Jelikož má navržená šifra proměnnou strukturu a bez znalosti klíče ji nelze odvodit, tak tyto metody není možné použít.
6. Nejpravděpodobnějším útokem na šifru se jeví útok na klíč nebo na heslo (pokud nebude zajištěno použití bezpečného hesla).
7. Šifra má dostatečně dlouhý klíč, aby odolala útokům postavených na redukci klíčů, alespoň za předpokladu současného výpočetního výkonu.
8. Nicméně, pro dokázání výše uvedeného by byla potřebná hlubší analýza. Například na vztahy mezi klíči a další speciální metody.

6 GRAFICKÉ UŽIVATELSKÉ ROZHRAŇÍ – GUI

V této části se budeme věnovat navrženému grafickému uživatelskému rozhraní (dále jen GUI) pro navrženou šifru, které uživateli zprostředkuje šifrování a dešifrování souborů. GUI je naprogramované opět v programovacím jazyce Python verze 3.x. Pro vytvoření GUI byla použita knihovna Tkinter, která umožňuje práci s Tcl/Tk. Python verze 3.x má vestavěnou verzi Tcl/Tk 8.5. Pro jednotlivé prvky (objekty) GUI byla použita podknihovna knihovny *Tkinter.ttk* – umožňující vytvářet tzv. „Tk themed widgets“ – tlačítka, pole s textem, pole pro zadání textu, políčka pro zaškrtnutí, pole pro volbu a další. Jednotlivé prvky GUI se vykreslují ve stylu podle operačního systému, na kterém GUI běží. To znamená, že vytvořené GUI bude vypadat jinak na platformě Microsoft Windows, jinak na Mac OS X a jinak na platformách Linuxu. [3,4] Vytvořené GUI se skládá z

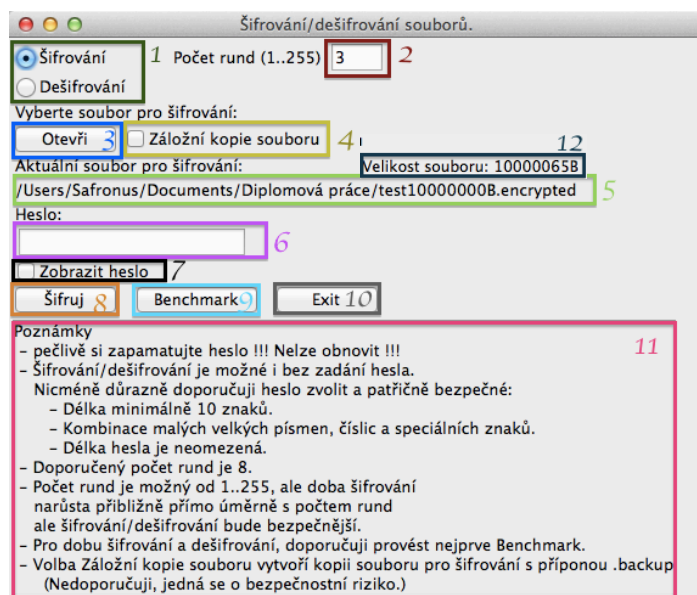
- hlavního okna
- podokno pro průběh šifrování/dešifrování
- podokno pro chybová hlášení
- podokno pro funkci benchmark

V dalších částech si tyto okna popíšeme. GUI bylo navrženo jako podpora testování navržené šifry, pro běžné používání by taky bylo dostačující, ale jinak by byla potřeba provést pár úprav. Nápomocné bylo jak z důvodu rychlejší volby souborů, volby operace šifrování/dešifrování, ale také díky funkci benchmark, která po distribuování v podobě souboru .exe umožnila testy na různých hardwarových konfiguracích.

Všechny zdrojové kódy jsou vypálené na přiloženém CD. Aplikace je velikosti a rozložením komponent optimalizována pro Mac OS X.

6.1 GUI – Hlavní okno

Hlavní okno je objekt *Tk()*. Pro seskupení jednotlivých prvků okna slouží objekt *Frame(rám)*, který se vytváří funkcí *ttk.Frame()*. Hlavní okno GUI se skládá ze 12 prvků a dalších popisků typu *Label*. Hlavní okno GUI vypadá následovně.



Obrázek 95 - GUI - Hlavní okno

Prvek 1 – Tlačítko slouží pro přepínání šifrování/dešifrování. V závislosti na stavu se mění popisky v hlavním okně a dále funkce pro šifrování/dešifrování.

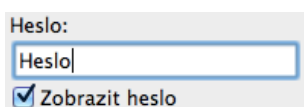
Prvek 2 – Jedná se o pole, do kterého se zadává počet rund. Pokud je zvoleno dešifrování, pole neaktivní (nejde volit počet rund), protože počet rund se určí ze zašifrovaného souboru.

Prvek 3 – Tlačítko, které otevře okno pro zvolení souboru pro šifrování/dešifrování. Využívá funkci `filedialog.askopenfile()`.

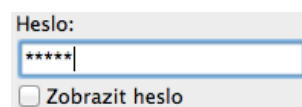
Prvek 4 – Zaškrtnutí tlačítko, které umožňuje volbu uložení záložní kopie při šifrování. Při šifrování totiž dochází k vymazání šifrovaného souboru. Pokud je zvoleno dešifrování, tlačítko je nefunkční, protože při dešifrování se vstupní soubor pro dešifrování nemaže.

Prvek 5 – Pole, které vypisuje cestu k souboru pro šifrování/dešifrování.

Prvek 6 – Pole, které umožňuje zadání hesla pro šifrování/dešifrování. Znak se defaultně zobrazují skrytě (znaky hvězdička – „*****“), ale po zaškrtnutí zaškrtnutí tlačítko níže (Prvek 7) se bude heslo zobrazovat neskrytě „heslo“.



Obrázek 96 - Zobrazení hesla



Obrázek 97 - Skrytí hesla

Prvek 7 – Zaškrťovací tlačítko, které umožňuje zobrazit/skrýt heslo v Prvku 6.

Prvek 8 – Tlačítko, které spustí průběh šifrování/dešifrování a otevře podokno s průběhem šifrování/dešifrování.

Prvek 9 – Tlačítko pro funkci benchmark, otevře podokno pro benchmark.

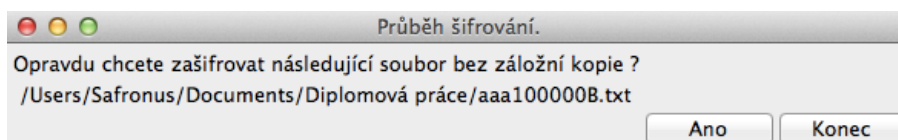
Prvek 10 – Tlačítko pro zavření hlavního okna a ukončení programu pro šifrování/dešifrování.

Prvek 11 – Tento prvek slouží sjednocení a pro výpis poznámek k programu pro šifrování/dešifrování.

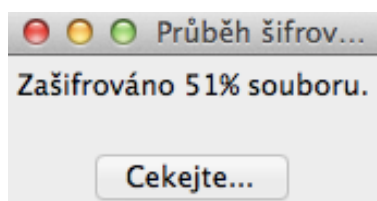
Prvek 12 – Pole pro výpis velikosti souboru pro šifrování/dešifrování

6.2 Podokno pro průběh šifrování/dešifrování

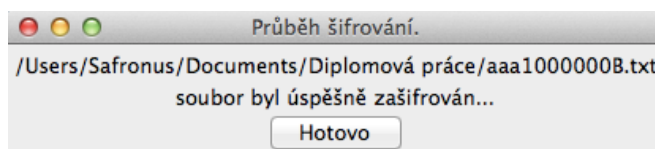
Okno je potomkem hlavního okna. Slouží pro potvrzení šifrování/dešifrování. A hlavně pro zobrazení průběhu šifrování/dešifrování aktuálního souboru. Nakonec pro potvrzení úspěšného šifrování/dešifrování.



Obrázek 98 - Podokno průběhu - potvrzení šifrování/dešifrování



Obrázek 99 - Podokno průběhu - aktuální stav šifrování/dešifrování

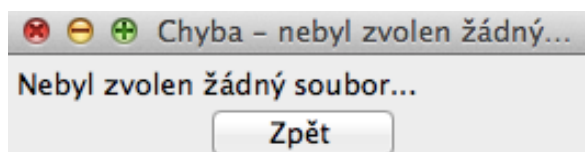


Obrázek 100 - Podokno průběhu - potvrzení o úspěšném šifrování/dešifrování

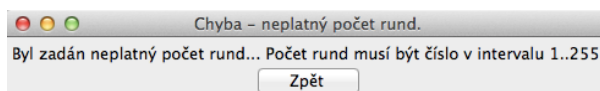
6.3 Podokno pro chybové hlášení

Podokno je opět potomkem hlavního okna. Slouží pro výpis chybových hlášení spojených s šifrováním a dešifrováním.

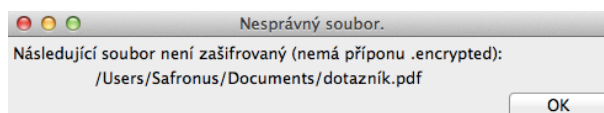
- Chybové hlášení při nezvolení souboru pro šifrování/dešifrování. (dále chybové hlášení 1)
- Chybové hlášení při zadání neplatného počtu rund. (dále chybové hlášení 2)
- Chybové hlášení pokud byl pokus o dešifrování souboru bez přípony .encrypted (dále chybové hlášení 3)
- Chybové hlášení pokud nebylo zadáno správné heslo pro dešifrování zvoleného zašifrovaného souboru. (dále chybové hlášení 4)



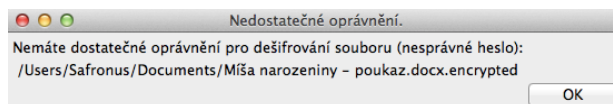
Obrázek 101 – chybové hlášení 1



Obrázek 102 – chybové hlášení 2



Obrázek 103 – chybové hlášení 3



Obrázek 104 – chybové hlášení 4

6.4 Podokno pro funkci benchmark

Vzhled okna je ukázán na obrázku č. 65 v kapitole 4.7. Okno slouží pro výpis výsledků z funkce pro benchmark, která provede otestování rychlosti v závislosti na hardwarové konfiguraci.

Výsledky byly nápomocné zejména při analýze rychlosti a výpočetní složitosti navržené šifry. Podobné fungování je popsáno opět v kapitole 4.7.

6.5 Možnosti zlepšení a rozšíření funkcí GUI

1. Jako první by se jednalo o implementaci možnosti použití souboru jako vstupního hesla nebo jiného způsobu, jak dosáhnout bezpečné vstupní heslo. Například generování náhodných souborů pohybem kurzoru myši. Nebo volby hesla za uživatele a jeho vygenerování do souboru.
2. Dále přidání možnosti šifrovat více souborů naráz nebo celé složky.
3. Při volbě souboru provedení výpočtu přibližné doby šifrování/dešifrování na základě velikosti a krátkého benchmarku.
4. Implementaci aplikace aby běžela na více jádrech procesorů, ne jen na jednom.
5. Samozřejmě předělání implementace zdrojových kódů.

7 ZHODNOCENÍ VYUŽITELNOSTI NAVRŽENÉ ŠIFRY PRO MOBILNÍ ZAŘÍZENÍ

Využitelnost navržené šifry pro mobilní zařízení se odvíjí od následujících věcí.

1. Od implementace šifry – Programovací jazyk (problém přenosu například pro Python - Android využívá ořezanou verzi Javy), správa bloků (ukládání a načítání pouze bloků) nikoliv celého souboru a jiné.
2. Rychlost s jakou je šifra schopna šifrovat – výpočetní složitost

V mobilních zařízení se hlavně využívají proudové šifry k zašifrování komunikace. Ale v dnešní době smartphonů už není mobilní zařízení pouze pro komunikaci, ale je potřeba zabezpečovat soubory. Proto na řadu přicházejí blokové šifry, které dostávají nový prostor. Nicméně z důvodu platformy nebo programovacího prostředí mobilních zařízení může být v mnoha případech omezující, ne-li eliminující pro danou šifru. Dále stále mobilní zařízení nedosahující srovnatelné výpočetního výkonu na notebooky nebo PC stanice. I když vývoj jsem mílovými kroky kupředu.

Při návrhu šifry a pokusu o otestování na mobilním zařízení s Androidem jsem sám narazil na překážky. Dokonce takové, že otestování nebylo možné. Zejména z následujících důvodů:

1. Není možné distribuovat zdrojový kód programovacího jazyka Python verze 3.x na Android. Neexistuje nástroj, který by tento zdrojový kód převáděl do jazyka používaného na platformě Android. Možností do budoucna se zdá být navržené programovací prostředí pro vývoj skriptů a aplikací v jazyce Python verze 3 a vyšší, které se jmenuje QPython 3. Které už umožňuje spouštění zdrojových kódů na platformě Android, ale potrvá ještě dlouhou dobu, kdy budou dostupné knihovny pro tvorbu GUI, apod. Například ještě nyní nejsou dostupné všechny knihovny pro Python verze 3.x, které jsou dostupné pro Python verze 2.7.x. A to je asi největší kamenem úrazu. Možnost by spočívala v napsání šifry v jiném programovacím jazyce. Například za využití nástroje Xamarin – v jazyce C#.
2. Dalším problémem by mohla být rychlost navržené šifry. Šifra není až tak výkonná, jak by bylo potřeba. Na druhou stranu výpočetní výkon stále roste a pokud se nenajde bezpečnostní slabina navržené šifry, tak její použitelnost s poroste s růstem výpočetního výkonu techniky. Nicméně nárůstu výpočetního výkonu by šlo docílit zlepšení implementace.

7.1 Možnosti použití navržené šifry na mobilních zařízeních

Navržená bloková šifra se dá využít všude tam, kde není vyžadovaná příliš velká rychlost šifry. I když to se může změnit s kvalitou naprogramování. Takže například:

1. Pro šifrování malých souborů do velikosti 100kB, kdy nepoznáme rozdíl s jinými šiframi. Jak bylo uvedeno v kapitole věnované analýze výpočetní složitosti a rychlosti navržené šifry, tak pro mobilní zařízení můžeme odhadnout rychlost šifrování 10-20kB/s při použití jedné rundy.
2. Šifrování sms zpráv
3. Šifrování poznámek, kalendářů
4. E-Mailů
5. Za předpokladu důkazu bezpečnosti – šifrování klíčů pro rychlejší šifry – pro malé soubory není problém použití více rund.

Nicméně by bylo zajímavé šifru implementovat do jiného jazyka a otestovat vlastnosti přímo na mobilním zařízení. Plánuji implementování do programovacího jazyka C# přes prostředí Xamarin a otestování na zařízeních s Androidem, která jsou nejvíce rozšířená. Dále i na mobilech firmy Apple.

ZÁVĚR

Cílem této práce bylo navrhnout novou symetrickou šifru, zhodnotit její parametry bezpečnosti, rychlosti a porovnat ji s běžně používanými šiframi AES a 3DES (i DES, který se už nepoužívá). Dále zhodnotit využitelnost navržené šifry v mobilních zařízeních.

V teoretické části byla popsána problematika moderní kryptografie a principy šifer AES, DES a 3DES. Dále byly popsány základní principy a operace využívané v moderní kryptografii ve vztahu k blokovým šifrám. V druhé kapitole teoretické části byl uveden popis základních prvků a vlastností blokových šifer, které jsou nutné pro návrh nové šifry.

Praktické části je věnován největší úsek diplomové práce, kde byla nejprve popsána navržená šifra, důvody k volbě konkrétních metod, operací a prvků šifry. Dále je uveden popis principů navržených metod a ukázka zdrojového kódu všech prvků a části navržené šifry v jazyce Python. Navržená šifra má v sobě zahrnuté nové prvky, s jakými jsem se doposud u blokových šifer nesetkal. Jedná se o její flexibilní strukturu závislou na klíči. Šifra nemá jako ostatní šifry pevně daný sled operací pro šifrování. Prakticky většina prvků a operací šifry jsou řízeny klíčem pro šifrování, který je odvozen od uživatelského hesla pro šifrování souborů. Šifra má proměnlivou strukturu rundy, je schopná šifrovat v počtu rund v rozmezí 0..255, dále i režim činnosti a substituční tabulka se mění v závislosti na klíči. To znamená, že pokud neznáme klíč (heslo) pro šifrování, tak se šifra chová strukturovou průběhem šifrování nepředvídatelně. V šifře je použita operace sčítání modulo, která je základním stavebním prvkem šifry. Šifra se pokouší odolat všem běžně používaným kryptoanalytickým metodám. Šifra byla prioritně navržena s ohledem na kvalitu bezpečnosti, a proto byla zvolena délka klíče 512 bitů. Šifra se během analýzy výpočetní náročnosti a rychlosti ukázala jako pomalejší, než ostatní používané šifry AES, DES a 3DES. Pro implementaci šifry byl použit programovací jazyk Python verze 3.x, což se ukázalo nevýhoda kvůli rychlosti a přenositelnosti na jiné platformy. Při příštím vývoji bych se pravděpodobně jazyku Python vyhnul, i když doba naprogramování se za využití programovacího jazyka Python výrazně zredukovala. Pro obsluhu dané šifry bylo navrženo grafické uživatelské rozhraní, opět v jazyce Python, což pomohlo otestovat navrženou šifru ve všech aspektech.

Dvě největší části diplomové práce se věnovaly analýze výpočetní náročnosti, bezpečnosti navržené šifry, jejích prvků a operací použitých k šifrování/dešifrování.

V kapitole analýza výpočetní náročnosti byla otestovaná doba trvání šifrova-

cích/dešifrovacích operací a dalších operací použitých v šifře. Na základě pokusů byla odvozena výpočetní složitost použitých operací. Výsledky pokusu byly ukázány v grafech a pomocí tabulek. Pomocí navržené funkce, která slouží pro aproximaci rychlosti a doby trvání šifrování/dešifrování v závislosti na počtu rund a velikosti souboru, byly vlastnosti otestované na více zařízeních, což vedlo k posouzení vlivu hardwarové konfigurace na výkonnost šifry. Dále byla rychlost šifry porovnána s šiframi AES, DES a 3DES. A na konci je uvedena část popisující vylepšení výkonosti šifry. I když je rychlost šifry malá, je to cena za zvýšení bezpečnosti.

V druhé části, která se věnuje analýze bezpečnosti navržené šifry, byly posouzené všechny prvky a operace šifry z hlediska bezpečnosti. Jako možné bezpečnostní ohrožení lze uvést například použití slabého hesla (kratší než 10 znaků, které kombinuje čísla a písmena) pro šifrování a následné generování inicializačního vektoru. Umožnilo by to prolomení hesla slovníkovou metodou, které by znehodnotilo sílu klíče a celé šifry. Na základě toho zjištění bylo navrženo možné protiopatření zejména v možnosti použití obsahu souboru k šifrování. Dále byly prvky šifry otestované na vlastnost lavinový efekt, který je jedním z podstatných měřítek otestování bezpečnosti jednotlivých prvků šifry a ten teoreticky činí šifru odolnou proti lineární a diferenciální kryptoanalýze. Lavinový efekt byl otestován při změně vstupního bitu, při změně hesla pro šifrování, při generování klíče pro následující blok a dalších prvků šifry. Z výsledků této analýzy bylo v kapitole 5.2.1 zjištěno teoretické ohrožení bezpečnosti šifry, vůči které by bylo vhodné zavést protiopatření eliminující zjištěnou potenciální hrozbu. Hrozba spočívá v absenci lavinového efektu při změně vstupního bitu šifrovaného bloku, pokud jsou všechny předchozí bloky stejné. Lavinový efekt se při testech blížil k nule, což by teoreticky mohlo výrazně ulehčit prolomení šifry. Bylo zjištěno, že tento nedostatek je způsoben malou závislostí operací pro šifrování na hodnotě vstupních dat, kdy se k šifrování používal stejný klíč. V kapitole 5.2.1 byly popsány možná teoretická řešení, která by v tomto případě zajistily zvýšení lavinového efektu. Nicméně, posoudit jestli se jedná o vážnou hrozbu narušující bezpečnost šifry, by chtělo podrobnější analýzu, protože podle webových stránek[12] se v tomto ohledu šifra chová podobně jako šifra AES s režimem činnosti OFB a CFB, která jak je známo, dosud nebyla prolomena. Následně se posuzovalo použití kryptoanalytických metod na navrženou šifru a bezpečnost v porovnání s šiframi DES, 3DES a AES. Pokud shrneme výsledky, vyplývající z analýzy bezpečnosti navržené šifry, tak se šifra jeví jako odolná vůči běžně použitelným metodám kryptoanalýzy a metodám, které se používají s ohledem na strukturu šifry, protože navrže-

ná šifra nemá pevnou strukturu. Je však vhodné implementovat vylepšení z kapitoly 5.2.1, které zvýší bezpečnosti šifry a zajistí kvalitní lavinový efekt ve všech případech. Jedinou metodou použitelnou k prolomení šifry se jeví útok na klíč nebo heslo, protože chování šifry se silně odvíjí od klíče (hesla pro šifrování). Každopádně pro potvrzení nebo vyvrácení odolnosti by byla opět zapotřebí hlubší analýza bezpečnosti šifry.

V závěrečné kapitole je zhodnoceno využití navržené šifry pro mobilní zařízení. Fakt, že byla šifra navržená s ohledem na bezpečnost a ne na rychlost šifrování, může být omezujícím prvkem při použití na mobilních zařízeních. Nicméně na bezpečnosti bych neubíral, neboť vývoj techniky jde stále dopředu, což urychluje dobu šifrování navržené šifry. Využít navrženou šifru jde v případech pro data o velikosti do 100kB, při stávající implementaci.

Pokud bychom chtěli šifru použít v praxi, bylo by nutné vylepšení šifry zejména v ohledu na zjištění v kapitole 5.2.1, nebo eliminovat možnost slabého hesla možností použít k šifrování jako heslo obsah souboru. Dále by bylo nutné šifru lépe implementovat pro zajištění vyšší rychlosti šifrování/dešifrování, zejména pokud bychom ji chtěli použít pro mobilní technologie.

Na závěr lze říct, že šifra dosahuje přijatelných výsledků a je vhodná pro budoucí vývoj.

SEZNAM POUŽITÉ LITERATURY

- [1] VONDRUŠKA, Pavel. Kryptologie, šifrování a tajná písma. 1. vyd. Praha: Albatros, 2006, 340 s. ISBN 80-00-01888-8.
- [2] SINGH, Simon. Kniha kódů a šifer: tajná komunikace od starého Egypta po kvantovou kryptografii. 2. vyd. v českém jazyce. Praha: Dokořán, 2009, 382 s. ISBN 978-80-7363-268-7.
- [3] Python 3.3.3 documentation [online]. 2014 [cit. 2014-02-05]. Dostupné z: <http://docs.python.org/3/>.
- [4] SUMMERFIELD, Mark. Python 3: Výukový kurz. Vyd. 1. Překlad Lukáš Krejčí. Brno: Computer Press, 2010, 584 s. ISBN 978-80-251-2737-7.
- [5] ŠENKEŘÍK, Roman. Přednášky z předmětu Kryptologie. 2005-2013.
- [6] <http://msdn.microsoft.com/en-us/library/system.security.cryptography.paddingmode%28v=vs.110%29.aspx>
- [7] <http://www.mvcr.cz/clanek/vyhlasaka-c-378-2006-sb-o-postupech-kvalifikovanych-poskytovatelu-certifikacnich-sluzeb.aspx>
- [8] http://www.cpubenchmark.net/low_end_cpus.html
- [9] http://www.androidbenchmark.net/cpumark_chart.html
- [10] SWENSON, Christopher. Modern cryptanalysis: techniques for advanced code breaking. Indianapolis: Wiley, c2008, xxviii, 236 s. ISBN 978-0-470-13593-8.
- [11] KHOVRTOVICH, Gaetan LEURENT a Christian LECHBERGER. Narrow Bicliques: Cryptanalysis of Full IDEA. Luxembourg, 2013. Vědecká práce. University of Luxembourg.
- [12] <http://aes.online-domain-tools.com/tool-form-submit/>

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

AES	Advanced Encryption Standard
ANSI	Inicializační vektor
CBC	Secure Hash Algorithm
CFB	Cipher Feedback Block
CPU	Central Processing Unit
DES	Data Encryption Standard
ECB	Electronic Code Book
EDE	Encryption-Decryption-Encryption
EEE	Encryption-Encryption-Encryption
FIPS	Federal Information Processing Standard
GHz	Gigahertz
GUI	Graphical User Interface
HDD	Hard Disc Drive
ISO	International Organization for Standartization
IV	Inicializační vektor
kB	kilobajt
MB	Megabajt
MHz	Megahertz
mod	Modulo
OFB	Output Feedback Block
PKC	Public Key Cryptography
PUB	Publication
RSA	Rivest-Shamir-Adleman – tvůrci
SSD	Solid State Disk

SEZNAM OBRÁZKŮ

<i>Obrázek 1 - Členění kryptografie [5]</i>	<i>14</i>
<i>Obrázek 2 - Blokové schéma principu symetrické kryptografie</i>	<i>15</i>
<i>Obrázek 3 – Zdrojový kód programu v jazyce Python verze 3.x pro ověření dešifrovací rovnice</i>	<i>20</i>
<i>Obrázek 4 - Výsledek běhu funkce pro ověření dešifrovací rovnice</i>	<i>20</i>
<i>Obrázek 5 - Režim činnosti ECB – Electronic Code Book [5]</i>	<i>22</i>
<i>Obrázek 6 - Režim činnosti CBC - Cipher Block Chaining [5]</i>	<i>23</i>
<i>Obrázek 7 - Režim činnosti OFB - Output Feedback Block [5]</i>	<i>24</i>
<i>Obrázek 8 - Režim činnosti CFB - Cipher Feedback Block [5]</i>	<i>24</i>
<i>Obrázek 9 - Blokové schéma šifry DES [5]</i>	<i>25</i>
<i>Obrázek 10 - Schéma jedné rundy [5]</i>	<i>25</i>
<i>Obrázek 11 - Průběh šifry 3DES [5]</i>	<i>26</i>
<i>Obrázek 12 - AES - ByteSub Transformation [5]</i>	<i>27</i>
<i>Obrázek 13 - Struktura S-Boxu [5]</i>	<i>27</i>
<i>Obrázek 14 - AES - ShiftRow Transformation [5]</i>	<i>28</i>
<i>Obrázek 15 - AES - MixColumn Transformation [5]</i>	<i>28</i>
<i>Obrázek 16 - AES - Add Round Key [5]</i>	<i>28</i>
<i>Obrázek 17 - AES – šifrování [5]</i>	<i>29</i>
<i>Obrázek 18 - AES – dešifrování [5]</i>	<i>29</i>
<i>Obrázek 19 – Aproximace vztahu mezi bezpečností a rychlostí šifry</i>	<i>31</i>
<i>Obrázek 20 - Blokové schéma šifrování celého bloku dat</i>	<i>37</i>
<i>Obrázek 21 - Blokové schéma šifrování necelého bloku dat</i>	<i>38</i>
<i>Obrázek 22 - Blokové schéma průběhu šifrování souboru</i>	<i>39</i>
<i>Obrázek 23 - Blokové schéma dešifrování celého bloku dat</i>	<i>40</i>
<i>Obrázek 24 - Blokové schéma dešifrování necelého bloku dat</i>	<i>41</i>
<i>Obrázek 25 - Blokové schéma průběhu dešifrování souboru</i>	<i>42</i>
<i>Obrázek 26 - Blokové schéma průběhu generování IV a hashe IV</i>	<i>43</i>
<i>Obrázek 27 - Defaultní řetězec v jazyce Python</i>	<i>43</i>
<i>Obrázek 28 - Zdrojový kód pro Krok 5</i>	<i>47</i>
<i>Obrázek 29 - Zdrojový kód funkce pro generování mapy indexů</i>	<i>48</i>
<i>Obrázek 30 - Zdrojový kód funkce pro zamíchání substituční tabulky</i>	<i>49</i>
<i>Obrázek 31 - Zdrojový kód funkce pro substituci</i>	<i>50</i>

<i>Obrázek 32 - Zdrojový kód funkce Add - přičtení klíče</i>	<i>51</i>
<i>Obrázek 33 - Zdrojový kód funkce pro mixování.....</i>	<i>52</i>
<i>Obrázek 34 - Zdrojový kód funkce pro inverzní operaci k substituci</i>	<i>53</i>
<i>Obrázek 35 - Zdrojový kód funkce pro inverzní operaci k Add - odečtení klíče.....</i>	<i>54</i>
<i>Obrázek 36 - Zdrojový kód funkce pro inverzní operaci k mixování.....</i>	<i>54</i>
<i>Obrázek 37 - Zdrojový kód funkce pro vygenerování posloupnosti operací</i>	<i>57</i>
<i>Obrázek 38 – Vývojový diagram pro určení hodnoty i-tého bajtu nového klíče</i>	<i>58</i>
<i>Obrázek 39 - Zdrojový kód funkce pro posunutí klíče</i>	<i>58</i>
<i>Obrázek 40 - zdrojový kód funkce pro zakódování bajtu udávající počet doplněných znaků.....</i>	<i>60</i>
<i>Obrázek 41 - zdrojový kód funkce pro dekodování bajtu udávající počet doplněných znaků.....</i>	<i>61</i>
<i>Obrázek 42 - zdrojový kód funkce pro zakódování počtu rund</i>	<i>61</i>
<i>Obrázek 43 - zdrojový kód funkce pro dekodování počtu rund</i>	<i>62</i>
<i>Obrázek 44 – Graf času při substituci pro náhodně generovaný vstupní blok a klíč.....</i>	<i>65</i>
<i>Obrázek 45 - Graf závislosti času pro substituci na počtu vstupních bloků (rund).....</i>	<i>66</i>
<i>Obrázek 46 - Graf času pro přičtení klíče pro náhodně generovaný klíč</i>	<i>68</i>
<i>Obrázek 47 – Graf počtu operací přičtení klíče pro náhodně generovaný klíč</i>	<i>68</i>
<i>Obrázek 48 - Graf závislosti času pro přičtení klíče na počtu vstupních bloků (rund)</i>	<i>69</i>
<i>Obrázek 49 - Graf času při mixování pro náhodně generovaný vstupní blok a klíč</i>	<i>71</i>
<i>Obrázek 50 - Graf závislosti času pro mixování na počtu vstupních bloků (rund)</i>	<i>71</i>
<i>Obrázek 51 - Graf času při inverzní operaci k substituci pro náhodně generovaný vstup a klíč.....</i>	<i>73</i>
<i>Obrázek 52 - Graf závislosti času pro inverzní operaci k substituci na počtu vstupních bloků (rund).....</i>	<i>73</i>
<i>Obrázek 53 - Graf času při odečtení klíče pro náhodné data a klíč.....</i>	<i>75</i>
<i>Obrázek 54 - Graf závislosti času pro odečtení klíče na počtu vstupních bloků (rund).....</i>	<i>76</i>
<i>Obrázek 55 - Graf času při inverzní operaci k mixování pro náhodný vstup.....</i>	<i>77</i>
<i>Obrázek 56 - Graf závislosti času pro inverzní operaci k mixování na počtu vstupních bloků (rund).....</i>	<i>78</i>
<i>Obrázek 57 - Grafu času pro operaci zamíchání substituční tabulky pro náhodně generovaný klíč.....</i>	<i>80</i>

<i>Obrázek 58 - Graf závislosti času pro operaci zamíchání substituční tabulky na počtu šifrovaných/dešifrovaných bloků</i>	81
<i>Obrázek 59 - Graf času pro operaci posunutí klíče pro náhodný vstup</i>	82
<i>Obrázek 60 - Graf závislosti při posunutí klíče na počtu šifrovaných/dešifrovaných bloků</i>	83
<i>Obrázek 61 - Grafu času pro vygenerování posloupnosti operací pro 1 rundu</i>	84
<i>Obrázek 62 - Graf závislosti času pro vygenerování posloupnosti operací při šifrování/dešifrování na počtu bloků</i>	85
<i>Obrázek 63 - Graf času pro vygenerování IV z hesla o délce 1 bajt</i>	86
<i>Obrázek 64 - Graf závislosti času generování IV z hesla o délce n bajtů (znaků)</i>	87
<i>Obrázek 65 - Výsledek funkce pro benchmark</i>	91
<i>Obrázek 66 - Výsledek benchmarku pro hardwarovou konfiguraci č. 1</i> <i>Obrázek 67 - Výsledek benchmarku pro hardwarovou konfiguraci č. 2</i>	92
<i>Obrázek 68 - Výsledek benchmarku pro hardwarovou konfiguraci č. 3</i>	93
<i>Obrázek 69 - Výsledek benchmarku pro hardwarovou konfiguraci č. 4</i>	93
<i>Obrázek 70 - Výsledek benchmarku pro hardwarovou konfiguraci č. 5</i>	93
<i>Obrázek 71 - Výsledek benchmarku pro hardwarovou konfiguraci č. 6</i>	93
<i>Obrázek 72 - Výsledek benchmarku pro hardwarovou konfiguraci č. 7</i>	93
<i>Obrázek 73 - Výsledek benchmarku pro hardwarovou konfiguraci č. 8</i>	93
<i>Obrázek 74 - Výsledek benchmarku pro hardwarovou konfiguraci č. 9</i>	94
<i>Obrázek 75 - Výsledek benchmarku pro hardwarovou konfiguraci č. 10</i>	94
<i>Obrázek 76 - Výsledek benchmarku pro hardwarovou konfiguraci č. 11</i>	94
<i>Obrázek 77 - Výsledek benchmarku pro hardwarovou konfiguraci č. 12</i>	94
<i>Obrázek 78 - Výsledek benchmarku pro hardwarovou konfiguraci č. 13</i>	94
<i>Obrázek 79 - Výsledek benchmarku pro hardwarovou konfiguraci č. 14</i>	94
<i>Obrázek 80 - Výsledek benchmarku pro hardwarovou konfiguraci č. 15</i>	95
<i>Obrázek 81 - Výsledek benchmarku pro hardwarovou konfiguraci č. 16</i>	95
<i>Obrázek 82 - Výsledek benchmarku pro hardwarovou konfiguraci č. 17</i>	95
<i>Obrázek 83 - Výsledek benchmarku pro hardwarovou konfiguraci č. 18</i>	95
<i>Obrázek 84 - Výsledek benchmarku pro hardwarovou konfiguraci č. 19</i>	95
<i>Obrázek 85 - Výsledek benchmarku pro hardwarovou konfiguraci č. 20</i>	95
<i>Obrázek 86 - Výsledek benchmarku pro procesor Intel Pentium 4 2.66 GHz programem PassMark [8]</i>	97

<i>Obrázek 87 - Část tabulky benchmarku procesorů zařízení s Androidem [9]</i>	<i>97</i>
<i>Obrázek 88 - Graf výsledků doby šifrování šifer AES, DES, 3DES a navržené šifry</i>	<i>98</i>
<i>Obrázek 89 - Zdrojový kód pro výpočet počtu režimů činnosti pro jeden bajt klíče a jeho výsledek.....</i>	<i>105</i>
<i>Obrázek 90 - Zdrojový kód funkce pro provázání změn vstupních dat.....</i>	<i>114</i>
<i>Obrázek 91 - Zdrojový kód funkce pro výpočet lavinového efektu</i>	<i>118</i>
<i>Obrázek 92 - Četnost znaků zašifrovaného souboru na základě frekvenční analýzy - navržená šifra</i>	<i>119</i>
<i>Obrázek 93 - Četnost znaků zašifrovaného souboru na základě frekvenční analýzy – AES-256.....</i>	<i>119</i>
<i>Obrázek 94 - Zdrojový kód funkce pro výpočet četnosti znaků zašifrovaného textu</i>	<i>120</i>
<i>Obrázek 95 - GUI - Hlavní okno</i>	<i>126</i>
<i>Obrázek 96 - Zobrazení hesla</i>	<i>126</i>
<i>Obrázek 97 - Skrytí hesla</i>	<i>126</i>
<i>Obrázek 98 - Podokno průběhu - potvrzení šifrování/dešifrování</i>	<i>127</i>
<i>Obrázek 99 - Podokno průběhu - aktuální stav šifrování/dešifrování.....</i>	<i>127</i>
<i>Obrázek 100 - Podokno průběhu - potvrzení o úspěšném šifrování/dešifrování</i>	<i>128</i>
<i>Obrázek 101 – chybové hlášení 1</i>	<i>128</i>
<i>Obrázek 102 – chybové hlášení 2</i>	<i>128</i>
<i>Obrázek 103 – chybové hlášení 3</i>	<i>128</i>
<i>Obrázek 104 – chybové hlášení 4</i>	<i>129</i>

SEZNAM TABULEK

<i>Tabulka 1 - Operace XOR</i>	<i>16</i>
<i>Tabulka 2 - Přiřazení posloupnosti pro i-tou rundu</i>	<i>56</i>
<i>Tabulka 3 - Tabulka srovnání operací pro šifrování a dešifrování</i>	<i>79</i>
<i>Tabulka 4 - Časy pro šifrování a dešifrování souborů v závislosti na obsahu a velikosti</i>	<i>89</i>
<i>Tabulka 5 – Hardwarové konfigurace pro výsledky benchmarků</i>	<i>92</i>
<i>Tabulka 6 - Výsledné doby šifrování pro šifry AES, DES a 3DES v porovnání s navrženou šifrou</i>	<i>98</i>
<i>Tabulka 7 - Počet kombinací hesel v závislosti na délce a použitých znacích</i>	<i>101</i>

SEZNAM PŘÍLOH

PŘÍLOHA P I: OBSAH CD

PŘÍLOHA P II: SUBSTITUČNÍ TABULKA

**PŘÍLOHA P III: UKÁZKA NEZAŠIFROVANÉHO A
ZAŠIFROVANÉHO SOUBORU.**

PŘÍLOHA P I: OBSAH CD

Struktura obsahu přiloženého CD:

- Adresář **Text diplomové práce** – obsahuje text diplomové práce ve formátu PDF.
- Adresář **Zdrojové kódy** – obsahuje všechny zdrojové kódy s příponou .py programovacího jazyka Python verze 3.x potřebné ke spuštění navržené aplikace pro šifrování/dešifrování souborů za využití navržené šifry.
- Adresář **Spustitelná verze aplikace WIN** – obsahuje zkompilovanou verzi aplikace pro operační systém Microsoft Windows. Spouští se pomocí souboru **Main.exe**. Součástí kompilace je soubor pro otestování „**aaa1000B.txt**“ a soubory pro benchmarkovou funkci – „**generatedbenchfilebig.txt**“ a „**generatedbenchfilebig.txt.encrypted**“.

PŘÍLOHA P II: SUBSTITUČNÍ TABULKA

[149, 35, 170, 231, 92, 10, 164, 59, 66, 121, 182, 134, 80, 46, 115, 233, 26, 176, 95, 161, 51, 204, 125, 49, 209, 245, 169, 44, 91, 63, 74, 154, 118, 238, 87, 171, 4, 210, 41, 19, 140, 196, 162, 20, 218, 11, 112, 232, 27, 68, 251, 175, 15, 90, 21, 222, 25, 22, 248, 234, 142, 109, 94, 160, 122, 104, 32, 172, 151, 192, 126, 157, 135, 237, 131, 235, 141, 130, 3, , 243, 99, 100, 168, 52, 207, 0, 79, 103, 97, 62, 1, 120, 73, 249, 70, 113, 38, 200, 254, 179, 110, 43, 186, 253, 124, 246, 40, 47, 145, 183, 76, , 133, 16, 116, 208, 158, 143, 195, 144, 24, 225, 31, 181, 30, 29, 54, 205, 101, 114, 197, 89, 255, 81, 227, 78, 28, 226, 106, 193, 107, 50, 33, 2, 8, 239, 212, 173, 111, 163, 223, 220, 86, 241, 190, 132, 146, 180, 57, 240, 166, 217, 72, 71, 228, 156, 42, 185, 102, 139, 65, 77, 9, 198, 199, 34, 224, 221, 13, 187, 88, 117, 83, 136, 5, 236, 6, 165, 64, 247, 98, 84, 216, 119, 129, 82, 250, 152, 242, 184, 96, 202, 201, 229, 108, 93, 61, 17, 138, 153, 174, 60, 23, 18, 148, 75, 85, 244, 123, 56, 167, 48, 147, 189, 230, 159, 137, 194, 214, 203, 55, 39, 211, 128, 219, 215, 188, 53, 14, 37, 36, 69, 150, 213, 155, 178, 58, 12, 177, 45, 7, 67, 206, 127, 105]

PŘÍLOHA P III: UKÁZKA NEZAŠIFROVANÉHO A ZAŠIFROVANÉHO SOUBORU.

