Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Disertační práce

Návrh polymorfních struktur v symetrické kryptografii

Design of the Polymorphous Structures in the Symmetric Cryptography

Autor: **Ing. Petr Žáček**

Studijní program: Inženýrská informatika (P3902)
Studijní obor: Inženýrská informatika (3902V023)

Školitel: prof. Mgr. Roman Jašek, Ph.D., DBA
Konzultant: Ing. David Malaník, Ph.D.

Zlín, Květen 2021

v edici **disertační práce**.

Publikace byla vydána v roce 2021.

Klíčová slova: *kryptologie, symetrická kryptografie, kryptografický systém, blokové šifra, režim činnosti blokových šifer, šifrovací funkce, runda, délka bloku, délka klíče, polymorfismus*.

Key words: *cryptology, symmetric-key cryptography, cryptography system, block cipher, block cipher mode of operation, encryption function, round, block length, key length, polymorphism*.

Práce je dostupná v Knihovně UTB ve Zlíně.

Poděkování

Rád bych touto cestou poděkoval svému školiteli prof. Mgr. Romanovi Jaškovi, Ph.D., DBA nejen za poskytnutí odborné pomoci, ale i za poskytnutí možnosti být plnohodnotným kolegou a součástí Ústavu informatiky a umělé inteligence. Poděkování patří i za poskytnutí prostředků a profesně vědeckého zázemí, díky kterým jsem mohl rozvíjet své odborné zaměření na kryptologii, kybernetickou bezpečnost a testování softwaru.

Velké poděkování patří i konzultantovi, kolegovi a příteli Ing. Davidu Malaníkovi, Ph.D., a to nejen za odborné i formální konzultování v rámci doktorského studia, ale i za možnost být členem laboratoře pro kybernetickou bezpečnost a penetrační testování - PT LAB a mít možnost spolupodílet se na velmi zajímavých projektech.

Dále bych chtěl velmi poděkovat své drahé polovičce Anetě Kovářové za její neskutečnou trpělivost a toleranci, a která mě provází v rámci celého studia. V neposlední řadě bych rád poděkoval rodině, přátelům, kamarádům a všem zbývajícím za jejich trpělivost při poslechu popisu Triply Salted Smallie, za podporu tak i za cenné rady.

ABSTRAKT

Disertační práce se věnuje polymorfním strukturám v symetrické kryptografii. Součástí textu je přehled zabývající se symetrickou kryptografií blokových šifer a aktuálnímu stavu dané problematiky. Mezi hlavní cíle patří vymezení termínu polymorfních struktur v symetrické kryptografii, uvedení příkladů stávajících algoritmů a šifrovacích principů na základě vymezení.

Práce je dále souborem výsledků, kterých bylo dosaženo v rámci doktorského studia a navrhuje jednotný šifrovací systém založený na rozebraných principech s důrazem na polymorfní struktury.

Celkové řešení obsahuje i praktickou implementaci všech navržených struktur v komplexní polymorfní šifrovací systém s ukázkou fungování. Práce dále prezentuje zhodnocení kvality návrhu včetně otestování systému.

ABSTRACT

The dissertation is focused on polymorphous structures in symmetric cryptography. The text includes an overview of symmetric cryptography of block ciphers and the actual state of that field. The main objectives are definition of the term polymorphic structures in symmetric cryptography, introducing examples of existing algorithms and principles based on definition.

The work is further a set of results achieved under Doctoral Study and proposes a single encryption system based on describes and designed principles with emphasis to the polymorphic structures.

The overal solution also includes the practical show of implementation of all proposed structures in a comprehensive polymorphous cryptographic system with a demonstration of functioning. The work further presents an evaluation of the design quality, including the testing of the system.

OBSAH

ÚVOD

V dnešní době není symetrická kryptografie, konkrétně blokové šifry v zájmu pozornosti IT specialistů. Oblast symetrické kryptografie od roku 2005 neprochází výrazným vývojem v porovnání s jinými oblastmi. Bloková šifra AES vytvořená v roce 1988 je dominantní šifrou přes 20 let a je používána v drtivé většině aplikací. Šifře AES může v současnosti konkurovat jen několik standardizovaných zástupců blokových šifer jako například Camellia, Kuznyechik. Doba využívání algoritmů by se na jednu stranu mohla jevit jako hrozba z důvodu dostatku času pro nalezení slabiny k prolomení, ale na druhou stranu, pokud za tento čas nebyla nalezen významný problém v algoritmu, tak by to mohlo znamenat kvalitu algoritmu. Dlouhodobá aplikace několika šifer otevírá dveře k vývoji nových algoritmů, a to zejména z důvodu dostupnosti alternativního řešení při potenciálním narušení bezpečnosti stávajících algoritmů. [1]; [2]; [3]

Většina hlavních zástupců blokových šifer (viz kapitola č. 4) má společnou jednu vlastnost, a to statické algoritmy. Jejich bezpečnost je postavena výhradně na vstupním klíči, případně na vstupních datech v závislosti na situaci použití režimů činnosti. Princip činnosti, struktura a další parametry zůstávají během celého procesu šifrování neměnné. Existuje však několik výjimek mezi zástupci blokových šifer, které bychom mohli označit za polymorfní. Jedná se hlavně o šifry, které mají proměnlivou délku šifrovaných bloků [4]; [5], dále o tzv. „tweakable“ [6]; [7]; [8] blokové šifry anebo šifry s klíčově závislým odvozením substitučních tabulek, permutačních boxů, nebo délkou klíče. [10]; [11] Na poli blokových šifer se vyskytují i šifry kombinující více vlastností najednou, nicméně v současné době (k datu 11. květen 2021) neexistuje žádná veřejně dostupná šifra, která by kombinovala vlastnosti všechny.

Výše uvedené důvody posloužily jako hlavní motivace pro tvorbu nové polymorfního kryptografického systému, který by byl vhodnou alternativou ke stávajícím šifrám/systémům. Hlavním cílem disertační práce byla tvorba kryptografického systému, u kterého by nebylo předem možné určit strukturu šifrování nebo její parametry bez znalosti vstupních dat nebo nastavení uživatele → šifrovacího klíče. Systému, kde by všechny aspekty byly založeny na závislostech jednotlivých části odvíjejících se od vstupních dat, parametrů a klíče. Navíc kde by se i proces šifrování a parametry pro šifrování následujících bloků odvíjel od aktuálního průběhu.

Výsledný návrh rozepsaný v disertační práci tyto vlastnosti splňuje a rozšiřuje o využití náhodných dat, kdy dochází ke změně systému v závislosti na jeho spuštění. A to bez nutnost změny uživatelského vstupu. Výsledný návrh

funguje tak, že bez znalosti vstupního klíče, nastavení vstupních parametrů či aktuálního stavu systému a vstupní entropie (náhodnosti), nelze predikovat parametry, rozložení operací, průběh operací a použitý klíč nad aktuálně šifrovaným blokem dat.

Je nutné dodat, že náplň disertační práce navazuje na mou vlastní výzkumnou diplomovou práci z roku 2014, přičemž tam vytvořenou šifru inovativně rozšiřuje. Diplomová práce byla sepsána na téma „Tvorba nové symetrické šifry pro mobilní zařízení“. [11] V této práci byla vytvořena šifra, která by sice mohla být označena za polymorfní, nicméně pro její plné funkční nasazení by bylo nutno realizovat další řadu vylepšení zahrnující vytvoření polymorfních aspektů všech částí, parametrů a operací. Touto problematikou se zabývá disertační práce, která rozšiřuje všechny zmíněné aspekty a vytváří výsledný komplexní kryptografický systém s polymorfními vlastnostmi.

1 CÍLE DISERTAČNÍ PRÁCE

Hlavním cílem disertační práce je výzkum v oblasti symetrické kryptografie blokových šifer a následný návrh vlastního komplexního kryptografického systému na základě tvorby jeho dílčích částí. Samotný návrh by měl být řešen v souladu s principem polymorfních struktur. To znamená volbu vhodných parametrů a částí kryptografického systému blokových šifer a jejich následnou modifikaci tak, aby chování výsledného systému bylo polymorfní. To celé při zachování základní filozofie kryptologie, že žádný algoritmus by neměl být tajný, ale jeho bezpečnost je současně matematicky podložená.

Součástí disertační práce bude také praktická implementace polymorfního kryptografického systému, který bude v co nejvyšší míře závislý na polymorfním chování – označený Smallie (ve variantě Triply Salted Smallie). Proto v rámci disertační práce bude vytvořených systém označen zkratkou TSS. Systém, který bude vytvořen z navržených jednotlivých části, jako výsledek dílčích cílů disertační práce. Bude provedena implementace a otestování v jazyce Python 3.x.

1.1 Dílčí cíle disertační práce

Cíle disertační práce lze rozdělit následovně:

1. Nastudování teoretického základu pro tvorbu blokových šifer a analýzu aktuálního stavu na poli polymorfních struktur v symetrické kryptografii.
2. Výběr parametrů, operací a vlastností blokových šifer vhodných pro polymorfizaci.
3. Návrh vlastního způsobu polymorfizace vybraných parametrů, operací a vlastností.
4. Praktická implementace polymorfních struktur v symetrické kryptografii (vlastní kryptografický systém TSS).
5. Analýza a případná optimalizace navržených a implementovaných polymorfních struktur z hlediska bezpečnosti.
6. Zhodnocení výsledků a popis budoucího výzkumu, či aplikaci v praxi.

2 ÚVOD DO ŘEŠENÉ PROBLEMATIKY

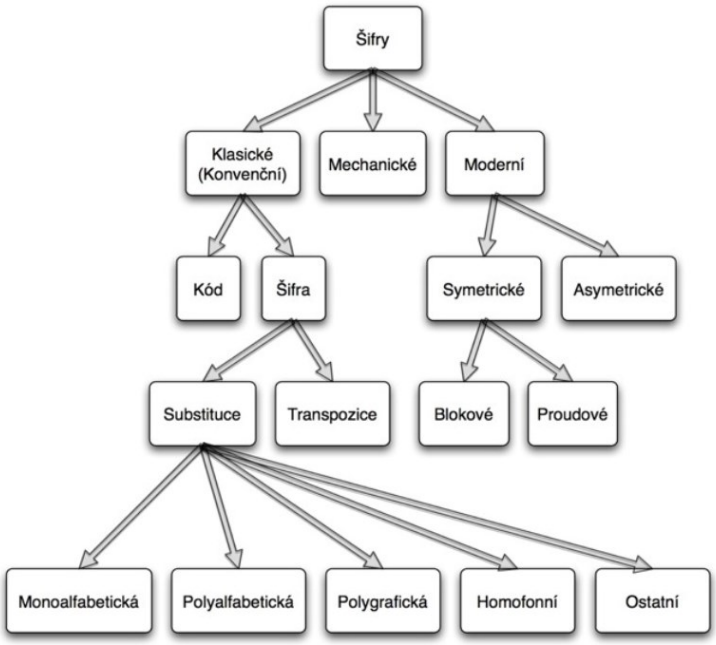
Tato část slouží jako teoretický a stručný úvod do problematiky disertační prací. Je v ní popsána řešená problematika, použitá terminologie a aktuální stav poznání. Teoretický základ je rozepsán tak, aby byla disertační práce vymezena a kategorizována v souladu s vědním oborem kryptologie a výsledný navržený systém byl do něj patřičně začleněn.

2.1 Kryptologie

Obecně se kryptologie považuje za rozsáhlé vědní odvětví, které se věnuje problematice návrhů, analýze a možnostem prolomení šifrovacích algoritmů. Kryptologie zahrnuje následující celky:

- Kryptografie – návrh a analýza navržených šifrovacích algoritmů
- Kryptoanalýza – analýza možnosti prolomení šifrovacích algoritmů
- Steganografie – návrh a analýza možností ukrytí informací

Tato práce se zaměřuje zejména na oblast kryptografie, nicméně částečně spadá i do kryptoanalýzy, protože návrh šifry je spojen s její s analýzou. Kryptografii dělíme do několika částí, jež můžeme vidět na uvedeném schématu (Obrázek 1).



Obrázek 1 – Základní rozdělení algoritmů v rámci kryptografie [12]

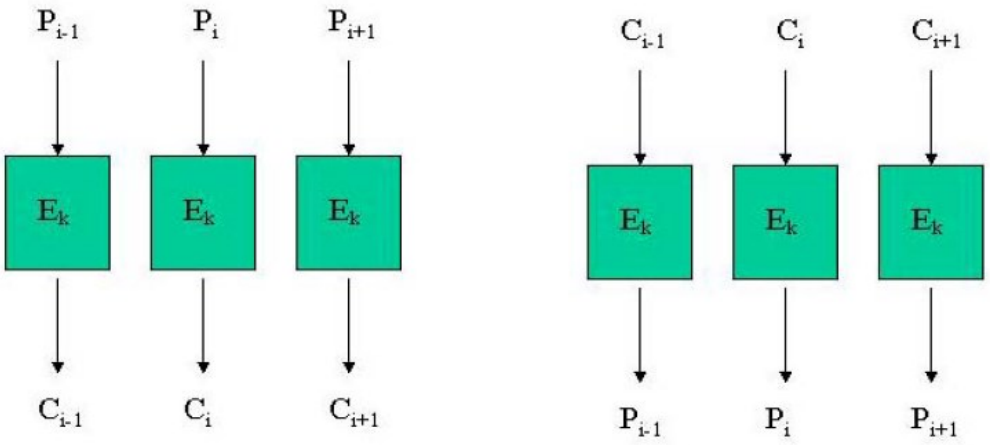
Základní dělení kryptologie je na klasickou (konvenční – papír/tužka) a moderní kryptografii (šifrování dat pomocí počítačů). Tato práce je zaměřena na moderní část, konkrétně symetrickou kryptografii s podmnožinou kryptografie blokových šifer.

2.2 Symetrická kryptografie

Jedná se tvorbu a analýzu algoritmů, které se vyznačují šifrováním i dešifrováním pomocí stejného klíče a liší se tím od asymetrické kryptografie, která pro šifrování využívá jiný klíč než pro dešifrování.

2.3 Kryptografie blokových šifer

Jak již bylo uvedeno, kryptografie blokových šifer je součástí moderní kryptografie, kdy během šifrování dat dochází k rozdělení na bloky o pevné délce. Data nejsou šifrována jako celek, ale na každý blok je aplikován šifrovací algoritmus, viz Obrázek 2. Délka je ve většině případů fixní, například 128, 192, 256 bitů.



Obrázek 2 – Režim činnosti blokových šifer ECB [12]

- P_i – aktuálně šifrovaný blok dat (vstup pro šifrovací algoritmus), nazývejme otevřená data
- C_i – aktuálně zašifrovaný blok (výstup šifrovacího algoritmu), nazývejme šifrová data
- E_k – algoritmus použitý pro zašifrování aktuálního bloku dat za pomocí klíče k

2.3.1 Režim činnosti blokových šifer

Principem funkce blokových šifer je rozdělení na bloky a jejich zašifrování. Problém nastává u opakujících se bloků, kdy je nutné zajistit jejich změnu a

úpravu před šifrováním. Tato operace je při šifrování nezbytná, neboť totožné bloky otevřeného textu by byly šifrovány podle stejného klíče a došlo by vrácení stejného výsledného šifrového textu. V praxi se tomuto stavu předchází pomocí režimu činnosti blokových šifer, který zajistí alternaci vstupních/výstupních dat způsobem, a tím zakryje fakt, že na vstupu byly stejné bloky dat. Problematika režimů činnosti využívaných v praxi je rozebrána v kapitole č 4.5.

2.3.2 Správa klíčů a počet rund

Při návrhu blokových šifer je potřeba počítat s vhodnou volbou počtu rund. Jedná se o opakované aplikování šifrovacích funkcí nad jedním blokem dat. Počet rund je součásti implementace konkrétní blokové šifry a je jedním ze základních vlastností, čím se blokové šifry od sebe liší.

S počtem rund souvisí i problematika rundovních klíčů – správa klíčů (anglicky key-management). Tento princip zajistí, že i když jsou operace šifrování aplikovány mnohokrát (podle počtu rund), tak že při každém aplikování šifrování je zvolen (odvozen či upraven) hlavní šifrovací klíč. Tudíž můžeme říci, že proces šifrování blokových šifer souvisí s určením hlavního klíče, který je v rámci key-managementu rozšířen či přepočítáván tak, aby každá runda využívala jiný klíč – nazývá se podklíč či rundovní klíč. Tento proces je opět stanoven v závislosti na blokové šifře a každá využívá jiný způsob. Jedná se o jeden ze zásadních faktorů, který ovlivňuje výslednou rychlost blokové šifry.

Platí následovné:

- Větší počet rund = delší průběh šifrování
- Komplexnější key-management = větší nároky na výpočetní kapacitu či paměť

V rámci výsledného systému TSS je správa klíčů řešená v režimu činnosti, který lze zařadit i mezi funkce pro odvození klíče. Nejsou zde řešeny klíče jako rundovní, ale jako klíče pro jednotlivé bloky. Počet rund je řešen v rámci kapitol 6.3.6 a 7.4.1 a souvisí i s parametrizací systému.

2.4 Základní pravidla při tvorbě kryptografických algoritmů

V rámci následující podkapitole stručně rozvedeme obecné principy při návrhu kryptografických algoritmů a šifer. Jsou to zejména otevřenost řešení, dále difúzi a konfúzi a v neposlední řadě rychlost šifrovacích algoritmů.

2.4.1 Otevřenost řešení

Důležitým principem při návrhu kryptografických algoritmů je tvorba v souladu s otevřeností řešení. Tím se myslí, že algoritmus by neměl být utajen,

ale měl by být postaven pouze na matematických principech. Principem je volba vhodných matematických komplexních/složitých funkcí pro zajištění bezpečnost, nikoliv zajištění bezpečnosti pomocí utajení algoritmu. Anglicky je tato problematika označována jako „security through the obscurity“. V rámci spousty historických šifer byla jejich bezpečnost zajištěna hlavně utajením algoritmu. Obdobně je bezpečnost řešena utajením v rámci tzv. „mystery caches“ ve hře Geocaching či jiných šifrovacích hrách. I v dnešní době by se dala najít spousta proprietárních řešení, která nelze považovat za špatná, ale rozhodně by jejich bezpečnost neměla spočívat pouze na utajení jejich principů.

2.4.2 Difúze a konfúze

Při návrhu šifer by mělo být vycházeno z principu dvou základních pravidel, které stanovil Claude Shannon v roce 1945. Jedná se o pravidla, které znatelným způsobem ztěžují možnost prolomení šifrového textu pouze s jeho znalostí. V rámci klasických šifer byla difúze i konfúze velmi slabá. [17]

Difúzi lze popsat jako princip při kterém je cílem co nejvíce rozprostřít statistické charakteristiky otevřeného textu (otevřených dat) do šifrového textu (šifrových dat). Jedná je tedy o vyjádření statistického vztahu mezi otevřenými daty a šifrovými daty. Zde je cílem, aby vztah mezi nimi byl co nejsložitější. V kontextu blokových šifer se jedná o využití režimů činnosti blokových šifer. Využití inicializačních vektorů či solí. Proces difúze dále souvisí s maximalizací entropie šifrových dat v porovnání s entropií otevřených dat. Například režim činnosti ECB má velmi malou míru difúze v porovnání s difúzí režimů CBC, GCM, XTS a dalších.

Můžeme difúzi rozdělit v souvislosti s rámci aktuálního bloku dat či v rámci celých dat. Cílem je dosáhnout alespoň velmi kvalitní difúze v rámci šifrování daného bloku dat, to znamená, aby při sebemenší změně otevřených dat daného bloku došlo k co možná nejvyšší změně šifrových dat daného bloku dat. Ideální difúze znamená je její rozšíření v rámci celých otevřených dat, to znamená že při sebemenší změně v otevřených datech by se změna měla co nejvíce projevit ve výsledných šifrových datech.

Konfúze lze popsat na statistické vyjádření vztahu klíče k šifrovému textu (šifrovým datům), kdy je cílem zajistit co nejvyšší míru konfúze. V praxi při návrhu blokových šifer je vhodná volba a návrh substitučních funkcí (S-Boxů). Například šifra AES má velmi vhodně navržené S-Boxy založené na problematice rovnic v galoidových polí stupně $2^8 \rightarrow (GF(2^8))$. [1]

Konfúze a difúze navrženého kryptografického systému TSS je rozebrána v kapitole 7.2 v rámci solení.

2.4.3 Rychlost šifrovacího algoritmu

V praxi je záměrem při tvorbě symetrických algoritmů (včetně blokových šifer) docílit toho, aby byl výsledný algoritmus dostatečně efektivní. Aby šifrování/dešifrování bylo dostatečně rychlé, a za využití co nejmenší výpočetní kapacity. To vše zároveň při zachování dostatečné bezpečnosti.

Navržený systém TSS byl navržen s důrazem na zajištění maximální bezpečnosti, nikoliv se zaměřením na efektivitu.

2.5 Problematika odvození šifrovacího klíče

Další částí kryptografie, které se věnuje disertační práce v rámci návrhu systému TSS je spojená s problematikou odvození klíče na základě tajných dat → např. z hesla, fráze, obsahu souboru a jiných. Jsou to algoritmy, které na základě tajných dat vytvoří šifrovací klíč požadované délky. Například, pokud uživatel chce šifrovat pomocí hesla, tak je heslo použito k odvození klíče a výsledkem jsou data závislá na hesle o konkrétní délce. Obdobně si lze proces představit jako hashování dat, kdy na vstupu mohou být libovolně dlouhá vstupní data a výsledkem je pevně daný klíč. Rozdílem je, že funkce pro odvození klíče je schopná generovat výstup libovolné délky dle potřeby dané šifry, pro kterou je potřeba vygenerovat klíč.

V praxi existuje mnoho algoritmů, které se používají k odvození klíčů na základě tajných dat. [21] Zde je výčet základních, které se využívají:

- Argon2 [18]
- crypt, bcrypt, scrypt [19]; [20]
- PBKDF2, HKDF [21]
- Vnitřní funkce šifrovacích algoritmů pro generování rundovních klíčů či podklíčů

Výše uvedené lze rozdělit do dvou skupin:

- Funkce pro odvození klíče (klíčů)
- Funkce pro posílení klíče
- Funkce pro rozšíření klíčů pro jednotlivé rundy/bloky

Funkce pro odvození klíče fungují na principu tvorby hlavního klíče na základě vstupních tajných dat ve formě hesla či odvození dalších klíčů z tajného klíče.

Funkce pro posílení klíče obvykle slouží pro posílení stávajícího klíče proti útoku hrubou silou. Posílení klíče ve smyslu zajištění vyšší entropie za dodržení obtížnosti odvození vstupních dat z klíče.

V rámci navržené kryptografického systému TSS bylo nutné vyřešit oba principy. Kdy při odvození klíče (klíčů) dochází v souvislostí se šifrování samostatných bloků. Je potřeba určit klíč pro šifrování prvního bloku a pak následná modifikace klíče vždy před šifrováním dalšího bloku dat.

Problematika tvorby hlavního klíče je řešena v kapitole 7.3. V rámci práce je označován jako inicializační vektor (*IV*).

3 TERMINOLOGIE A VYMEZENÍ POJMŮ DISERTAČNÍ PRÁCE

V rámci této disertační práce bude využita všeobecně zažitá terminologie v kontextu s kryptografií blokových šifer. Zároveň je využita terminologie, která je specificky uzpůsobena potřebám realizace polymorfního kryptografického systému TSS. V důsledku následující podkapitoly slouží jako seznam termínu a pojmů, které byly využity v souvislosti s návrhem polymorfního kryptografického systému TSS. Následující podkapitoly slouží pro lepší orientaci a přehlednost v rámci textu disertační práce.

Vymezení je dále podrobněji zachyceno v dalších podkapitolách.

3.1 Otevřená data

Jedná se o data od dané velikosti vyjádřenou v bajtech. Data, které jsou šifrovány a jsou tedy vstupem blokové šifry. V konkrétní aplikaci v rámci disertační práce a v rámci systému TSS se jedná o následující položky:

- Otevřená uživatelská data – vstup pro zašifrování zvolený uživatelem
- Sůl pro „předsolení“ – viz kapitoly 7.2.2 a 7.4.4
- Sůl pro „zasolení“ – viz kapitoly 7.2.3 a 7.4.4

V praxi se jedná například o obsah souboru k šifrování nebo text, dále např. data obsažená v paketu při komunikaci přes internet před šifrováním. Data jsou obvykle doplněná bajty dat v závislosti na délce bloků, kterou využívá daná bloková šifra. Procesu doplnění se říká padding.

3.1.1 Padding

Jako samostatnou krátkou kapitolu je vhodné vymezit a rozvést problematiku paddingu. Jedná se o doplnění poslední bloku dat, který je kratší než očekávaná délka bloku pro šifrování danou blokovou šifrou. V praxi je doplnění prováděno pomocí jednoho z následujících algoritmů:

- ANSI X9.23 – Doplnění nulových bajtů, kdy poslední bajt je binární reprezentací počtu doplněných bajtů [22]
- ISO 10126 – Doplnění o náhodné bajty, kdy poslední bajt je opět binární reprezentací počtu doplněných bajtů [23]
- PKCS#7 – Doplnění o patřičný počet bajtů, kdy každý z nich je reprezentován binární hodnotou počtu doplněných bajtů [24]

Z výše uvedených je v praxi pravděpodobně nejvíce využívaný způsob PCKS#7. Padding je v praxi spojen také s existencí několika útoků, které jsou ve

vazbě na režim činnosti CBC → Padding Oracle Attack a generalizace útoku POODLE. Nejen z toho důvodu byl kryptografický systém TSS navržen tak, aby nebylo nutné doplňování bloků dat a padding byl z procesu šifrování zcela eliminován.

3.2 Šifrová data

Jedná se o data, která jsou zejména výstupem po šifrování blokovou šifrou. Jejich délka je opět vyjádřena délkou v bajtech. V rámci realizace systému TSS je součástí i vnořená náhodně generovaná sůl použitá pro „nasolení“ inicializačního vektoru a finalizaci klíče pro šifrování prvního bloku otevřených dat. Konkrétní implementace a podrobnější popis viz kapitoly 7.2.4. a 7.2.5.

V praxi se většinou jedná o zašifrovaná data souboru nebo text.

3.3 Správa klíčů - šifrovací klíče

Jde o data vymezené délkou v bajtech, které využívá bloková šifra k zašifrování jednoho bloku dat. V rámci řešení se jedná nejprve o „nasolený“ inicializační vektor pro šifrování prvního bloku dat a následně o každý další klíč, který je pro šifrování každého dalšího bloku dat přepočítán v závislosti na předchozích stavech. Proto je zde v rámci textu budeme rozlišovat na 3 samostatné druhy klíčů.

- „nultý“ klíč → označený jako IV
- První klíč – slouží k zašifrování prvního bloku dat a nastavení procesu šifrování prvního bloku dat a modifikaci uživatelských tajných parametrů. Dále i k nastavení „předsolení“ a „zasolení“.
- Další klíče

3.3.1 Inicializační vektor

V praxi se jedná o náhodná data sloužící k zavedení difúze do šifrování a je většinou volen jako první hodnota (blok) v rámci režimu činnosti blokových šifer.

Pro potřeby návrhu systému byl využitý jiným způsobem. Zde je inicializační vektor transformací tajných uživatelsky zvolených klíčových dat v tzv. „nultý“ klíč. Pro odvození *IV* byla vytvořena nová vlastní funkce z kategorie funkcí pro odvození klíče. Funkce a princip je rozveden v rámci kapitoly 7.3. „Nultý“ klíč v systému TSS neslouží k šifrování (obdobně jako *IV* v praxi neslouží k šifrování), ale k modifikaci tajně uživatelsky zvolených parametrů pro potřeby „nasolení“ → délka nasolení a výpočet pozice soli v šifrových datech.

3.3.2 První klíč

Jedná se o „nasolený“ inicializační vektor pomocí náhodně generovaných dat → solí. Tento klíč už slouží k šifrování prvního bloku dat. Dále slouží k odvození dalších klíčů a k nastavení parametrů pro šifrování dalších bloků. Problematika je rozebrána podrobněji v rámci kapitoly 7.2.4.

V praxi lze tento klíč přirovnat k hlavnímu klíči či prvnímu klíči ze kterého jsou vypočteny rundovní klíče.

3.3.3 Další klíče

Jedná se o všechny další klíče sloužící k šifrování druhého a dalších bloků dat. Každý následující blok dat včetně parametrů šifrování je ovlivněn stavem předchozího klíče.

V praxi další klíče můžeme přirovnat k rundovním klíčům blokových šifer

3.4 Sůl

Jsou data s vysokou mírou entropie, která mají svou délku vyjádřenou v bajtech a jsou generovány náhodně (či pseudo-náhodně → odvíjí se od generátoru). Slouží pro zvýšení entropie a případné posílení vstupních dat, kde je entropie žádoucí. V rámci systému TSS jsou využité 3 nezávisle generované soli pro potřeby „předsolení“ – zanesení entropie (náhodnosti) do otevřených dat, „zasolení“ – zanesení entropie do otevřených dat a znejasnění délky otevřených dat a „nasolení“ – vnesení entropie do inicializačního vektoru a tím ovlivnění kvality prvního klíče. Podrobně je vše rozepsáno v kapitole 7.2.

V praxi se sůl používá například při odvození klíče z hesla, při hashování nebo také při posílení klíče.

3.5 Bloková šifra

Bez nadsázky je bloková šifra jádrem každého šifrovacího systému. Bloková šifra slouží k zašifrování jednoho bloku dat, tzn. transformace otevřených dat na šifrová pomocí vhodných funkcí. Podobně je bloková šifra systému TSS rozebrána v kapitole 7.1. Zároveň je ve svém původním tvaru rozebraná v rámci autorovi diplomové práci, kdy disertační práce blokovou šifru modifikuje.

V rámci systému TSS jsou řešeny operace:

- Substituce
- Transpozice
- Přičtení klíče

V praxi existuje spousta šifer kombinující různé druhy operací. Nejčastěji se jedná o

- Substituci – forma či aplikování substitučních boxů (S-Boxů) → nahrazení bajtu/bajtů dat otevřených dat za jiné
- Transpozice – aplikace permutačních boxů (P-Boxy) či permutace → změna pozice bajtů či bloků bajtů v otevřených datech
- Přičtení klíče – přičtení klíče (rundovních klíčů) k otevřeným datům za využití operací:
  - XOR
  - AND
  - OR
  - a dalších

Existující blokové šifry a ty které jsou využívané v praxi jsou uvedeny v kapitole 4. Blokové šifry se liší nejenom ve využitých operacích, ale v dalších následujících parametrech:

- Délka bloku
- Počet rund
- Tvorba podklíčů (rundovních klíčů)

Bloková šifra v rámci vytvořeného systému TSS, včetně všech operací a parametrů navržena tak, aby se chovala polymorfně.

3.6 Režim činnosti blokových šifer

Režimy činnosti používané v praxi včetně teoretického konceptu byly rozebrány v rámci kapitol 2.3.1 a 4.2. Jako součást systému TSS byla tato problematika řešena obdobně, ale opět se zaměřením na polymorfní struktury a chování. Režim činnosti systému TSS není řešen jako klasický režim činnosti, kdy dochází zejména k provázání otevřených dat se šifrovými daty, ale je řešen jako systém manipulující s klíči. Konkrétně se jedná o algoritmické odvození klíče pro šifrování dalšího bloku na základě:

- Parametrizace systému
- Náhodných dat ve formě solí → první klíč
- Předchozích otevřených dat
- Výstupních šifrových dat po zašifrování předchozích otevřených dat
- Aktuální klíče pro šifrování předchozích otevřených dat

Tento systém režimu činnosti v sobě kombinuje i odvození a správu podklíčů. Během doktorského studia bylo navržených hned několik režimů

činnosti a jejich konkrétní návrhy, včetně optimalizace a testů jsou rozebrány v kapitole 7.5.

3.7 Kryptografický systém

Tímto pojmem se rozumí komplexní systém, který ve vztahu s oblastí blokových šifer sestává nejen ze samotné šifry (blokové šifry), ale i z následujících částí:

- Bloková šifra – zajišťuje šifrování bloku dat
- Správa klíčů – odvození a výpočet všech potřebných klíčů k šifrování všech bloků dat
- Transformace tajných uživatelských dat v klíč či inicializační vektor
- Aplikace solí či nonce (šumu)
- Režim činnosti blokových šifer

Existuje mnoho kryptografických systémů, které kombinují různé šifry i různé režimy činností včetně dalších dílčích částí. Jde o systémy, které umožňují volbu šifer či dílčích algoritmů před šifrováním. Do kryptografického systému můžeme zařadit:

- protokol TLS – šifrování komunikace,
- šifrovací protokoly WEP, WPA či WPA2 – bezdrátový přenos Wi-Fi
- nástroj VeraCrypt či původní TrueCrypt - šifrování disků a souborů (multiplatformní),
- nástroj BitLocker – šifrování disků v rámci systému Windows,
- nástroj FileVault – šifrování disků v rámci systému Mac OS X,
- nástroj LUKS – šifrování v rámci systému Linux
- šifrování v rámci komprimace souborů s příponou .zip
- a další

4 SOUČASNÝ STAV ŘEŠENÉ PROBLEMATIKY

Následující kapitola je věnována současnému stavu problematiky kryptografii blokových šifer a struktur.

4.1 Fixní versus „polymorfní“

Tato část je zaměřena na rozlišení definic fixní a polymorfní šifry. Fixní šifra má nezávisle na vstupních datech (otevřená data a klíč) průběh šifrování a parametry neměnné, lze je tedy bez podrobnějších znalostí analyzovat. Obecná definice pro polymorfní šifry není v literatuře dohledatelná, což souvisí s tím, že samotná polymorfní šifra s velkou pravděpodobností neexistuje. Popsané jsou algoritmy tweakovací blokové šifry nebo jiné, které mění délku bloku nebo substituční tabulku, ale nejsou definovány jako polymorfní. Pro účely disertační práce je nezbytné tento rozdíl rámcově popsat a definovat.

Polymorfnost představuje vlastnost systému, která umožňuje systému měnit chování, vlastnosti, parametry či strukturu v čase a závislosti na okolních podmínkách či vstupních nebo aktuálních parametrech a nastaveních. Pokud tuto definici přeneseme do oblasti kryptografie, mluvíme o takových algoritmech (šifrách), u kterých se jedna či více vlastností mění na začátku nebo v průběhu šifrování. Jednou z možností je změna vstupního klíče nebo lépe inicializačního vektoru, která je následována změnou pořadí operací, délky bloku, či jiných parametrů nebo vlastností šifrovacího algoritmu.

Polymorfní šifra inklinuje nejvíce k takzvaným key-dependent (klíčové závislým) strukturám. Společným znakem těchto struktur je šifrovací klíč, který vstupuje jako skrytý (tajný) parametr do šifrování v symetrické kryptografii. Polymorfní šifru si lze představit jako sadu šifer s tisíci kombinacemi, kdy pro každou z nich bude jiný průběh šifrování, a právě klíč určí která z těch tisíci možností bude použita. Teoreticky bez znalosti klíče nelze určit, jaká z cest (šifer) byla použitá, což vede ke ztížení analýzy algoritmu. Tato skutečnost se jeví jako dvousečná zbraň, jelikož se na jednu stranu znesnadní identifikace slabých míst, na druhou stranu zkomplikování analýzy šifry vede k obtížné optimalizaci. Analýza a hledání slabých míst je jak pro kryptologa, tak pro útočníka stejně obtížná. Další přirovnání polymorfních šifer by mohlo být k principu „black-box“, které je založené na černé skříncе, kdy na vstupu máme parametry, klíč a vstupní data a bez nutnosti znalosti struktury uvnitř skříňky dostaneme výstupní zašifrovaná data na výstupu.

Z výše uvedené definice (popisu) můžeme odvodit, že blokové šifry můžeme rozdělit do následujících kategorií:

- Klasické šifry s fixní strukturou (většina algoritmů)
- Polymorfní šifry s proměnlivými strukturami

V následujících částech si tyto dvě kategorie rozebereme a ukážeme si jejich zástupce.

4.2 Běžné blokové šifry s fixní strukturou

Jak bylo popsáno v předchozí části, jedná se o šifry, u kterých jsou známé parametry a šifrovací funkce, včetně operací. Šifrovací klíč a vstupní data jsou jediné proměnné. U některých zástupců je možné na začátku zvolit délku klíče. U většiny zástupců volíme jako vstupní parametry režim činnosti spolu s paddingem. Sled operací, počet rund, délka klíče a zbylé parametry následně už zůstávají neměnné po zbytek šifrování.

Tento přístup je dlouhodobě používaný, má však své výhody a nevýhody.

Výhody:

- Pevná struktura a parametry vedou k možnosti podrobné analýzy
- Rychlost s využitím hardwaru vytvořeného na míru, instrukce procesorů pro šifru AES nebo paralelizace
- Nízká výpočetní náročnost díky absenci nutnosti manipulace s parametry a jejich přepočty
- Standardizovatelnost a možnost certifikace → v porovnáním s algoritmy s polymorfním chováním

Nevýhody:

- Nutnost paddingu (padding oracle útok [15])
- Nutnost režimu činnosti blokových šifer
- Teoretická nižší bezpečnost a větší možnost prolomení, z důvodu možnosti jednoho nastavení
- Možnost útoků cílených na strukturu → klasické kryptoanalytické metody či konkrétní Related-key (Boomerang switching) attack na šifru AES [31]

4.2.1 Zástupci mezi klasickými šiframi s fixní strukturou

V současné době je velká škála zástupců klasických blokových šifer s pevnou strukturou, každopádně jen pár jich je standardizovaných. Například organizace NIST standardizuje dvě využívané blokové šifry, a to AES a TDEA. [16] Mezi další standardizované šifry patří Camellia [2] (certifikace CRYPTREC – Japonsko a NESSIE – Evropská Unie) a šifra Kuznyechik [3] (Standard GOST

– Rusko). Existují i další šifry, které jsou hojně využívané i bez standardizace a mezi ně patří šifry Blowfish, Twofish, Threefish, Serpent, nebo IDEA. Dále pak nejnovější blokové šifry vytvořeny v rámci NSA – Simon, Speck, které se nevyznačují ničím novým, ale jsou výhradně navrženy pro oblast IoT [25] a spadají do skupiny tzv. „lehkých“ šifer, které lze využít na zařízením s méně výkonným HW. Do této skupiny spadají i další algoritmy jako TEA a její modifikace XTEA. [26]; [27] Dalšími hojně využívanými šiframi, které jsou proudové a lze je aplikovat jako blokové jsou Salsa20 a Chacha (obvykle v kombinaci s Poly-1305). Ve všech případech se jedná o blokové šifry s potvrzeným stupněm bezpečnosti bez známého aplikovatelného útoku či slabiny. Problematiku blokových šifer velmi dobře shrnuje kniha The Block Cipher Companion. [40]

Podrobnému výkladu se zde nebudeme věnovat, neboť šifry s fixní strukturou byly dostatečně popsány v mnoha literárních zdrojích, které jsou snadno dohledatelné online. V následující sekci budou popsány základní vlastnosti spolu s odkazem na podrobnější popis certifikovaných šifrovacích algoritmů. Vyjmenovaní zástupci a všeobecně užívané algoritmy nemají náznaky polymorfnosti.

4.2.2 Šifra AES

- Vznik – rok 1998
- Bloková šifra
- Délka bloku – 128 bitů
- Délka klíče – 128, 192 nebo 256 bitů
- Operace:
  - SubBytes – substituce bajtů dle substituční tabulky
  - ShiftRow – posun řádků o pevný počet bajtů
  - MixColumn – transpozice sloupců po čtyřech bajtech spolu s jejich vektorových součinem polynomiálního tvaru
  - AddRoundKey – přičtení rundovního klíče pomocí funkce XOR
- Generování rundovních klíčů
- Počet rund – 10, 12, 14 [1]

4.2.3 Šifra TDEA (dřívěji 3DES/TDES)

- Vznik – rok 1998 (princip šifrování 1975)
- Poslední modifikace 2017 (TDEA)
- Bloková šifra – feistelova struktura
- Délka bloku – 64 bitů
- Délka klíče - 168, 112 nebo 56 bitů

- Operace
- Substituce
- Permutace
- Počet rund – 16 [16]

4.2.4 Šifra Camellia

- Vznik – rok 2000 (MITSUBISHI)
- Bloková šifra – feistelova struktura
- Odvozena od šifer E2, MISTY1
- Délka bloku – 128 bitů
- Délka klíče – 128, 192 nebo 256 bitů
- Operace
- Substituce
- Generování podklíčů
- Odvození šifrovacích konstant
- FL a FL-1 funkce
- Počet rund – 18 nebo 24 [2]

4.2.5 Šifra Kuznyechik

- Vznik – rok 2015
- Bloková šifra – feistelova struktura
- Délka bloku – 128 bitů
- Délka klíče – 256 bitů
- Operace
- Substituce
- Permutace
- Počet rund – 10 [3]

4.3 Polymorfní šifry s proměnlivou strukturou (parametry)

Jako polymorfní šifry lze označit všechny šifry, které alespoň z části mění princip činnosti nebo parametry za běhu nebo aktuálního stavu. Šifer založených na tomto principu je málo, a ty co existují, nejsou všeobecně rozšířeny. V následující kapitole si uvedeme zástupce dle kategorií na základě části, která spadá do polymorfního chování.

4.3.1 Zástupci symetrické kryptografie s polymorfním chováním

Před samotným výpisem seznamu polymorfních šifer je nutné aktuální šifry rozdělit do kategorií. Kategorie se odvíjí od parametru, který funguje na polymorfním principu.

- **Blokové šifry založené na principu tweaku**

Prvním a nejvíce zmiňovaným zástupcem jsou šifry z kategorie tweakovacích šifer. Pojem tweakovací bloková šifra byl definován v roce 2001 autory Moses Liskov, Ronald L. Rivest a David Wagner. Dle autorů se jedná o takovou blokovou šifru, která má rozšířené fungování na základě tzv. „tweaku". „Tweak" je parametr, který kromě otevřených dat a klíče vstupuje do procesu šifrování. Prvním zástupcem a zároveň ukázkou se stala šifra DESX → rozšíření šifry DES o tweak. Autoři vymezili a definovali celou oblast tweakovacích šifer, včetně principu tvorby, a za jakých podmínek je pak můžeme označit za bezpečnou tweakovací šifru apod. [6]

Mezi zástupce tweakovacích blokových šifer patří [6]; [7]; [8]:

- DESX
- Deoxys-BC → rozšíření AES o tweak
- Kiasu-BC → jednoduchá modifikace AES-128 o tweak
- Joltik-BC → další úprava AES s rozšířením pro autentizaci
- SKINNY → skupina šifer

Tweak bychom mohli přirovnat částečně k parametru X, který je zmíněn v kapitole 7.4. Následně by se jednalo o tweak závislý na klíči jako číslo na intervalu $\langle 0;\ 255 \rangle$.

4.3.2 Blokové šifry s proměnlivou substituční tabulkou

Do této kategorie spadají šifry, které mají substituční tabulku odvozenou od klíče. Tyto šifry se označují jako „key-dependent S-Box ciphers". Mezi hlavní zástupce může zařadit následující blokové šifry:

- Blowfish [10]
- Khafre, Khufu [28]
- GOST [29]

Substituční tabulka je odvozena pouze jednou, a to na začátku.

4.3.3 Blokové šifry s proměnlivou délkou bloku

Jedná se o blokové šifry, které mohou šifrovat bloky dat s proměnlivou délkou. Tyto šifry se snaží eliminovat negativní faktor zahrnující nutnost použití režimu činnosti nebo paddingu (doplnění posledního bloku dat na patřičnou délku). [4]; [5]

4.3.4 Blokové šifry s více polymorfními parametry

Nejnověji navržená šifra DSDP splňuje nejvíce parametrů polymorfní šifry. Šifra má proměnlivou délku klíče s vlastním managementem klíčů, s proměnlivou substituční tabulku, proměnlivými permutačními boxy a proměnlivým počtem rund. [9]

4.4 Polymorfní kryptografické systémy a další návrhy

Na internetu lze dohledat další systémy, algoritmy či teoretické návrhy a vědecké publikace, které v menší či větší míře odpovídají myšlence polymorfních struktur [30]; [32]; [33]; [34], kdy poslední je obzvláště zajímavý článek z roku 2021 jako návrh polymorfizace šifry AES → P-AES. Ovšem těchto algoritmů či vědeckých článků není mnoho. Lze se domnívat, že systémů či algoritmů bude více, kdy jejich používání bude jako součást proprietárních systémech či tajné (viz systém TurboCrypt popsaný dále).

V další části si uvedeme dvě zajímavé existují řešení. První, už zmíněný, systém TurboCrypt, který v rámci své dokumentace vhodně popisuje základní myšlenku polymorfních šifer včetně s problematikou spojených základních výhod a problémů.

Druhým zástupcem (zástupci) je stránka makedonského autora Kostadina Bajalcalieva [49] → „Kostadin Bajalcaliev Cryptography Page“, který také popisuje myšlenku polymorfního šifrování s částečným matematických vyjádřením. Zároveň zde je možné vidět implementaci řešení. Stránka je zde uvedena hlavně z důvodu, že se jedná pravděpodobně o první dokumentaci myšlenky polymorfního šifrování. Stránka popisuje výzkum od roku 1995 do roku 2000.

4.4.1 Kryptografický systém TurboCrypt

Zajímavým systémem, který lze dohledat je TurboCrypt. Jedná se o kryptografický systém z roku 2008, který je principem velmi podobný systémům TrueCrypt a posléze VeraCrypt. [35]; [36]

Jak se je možné dočíst v dokumentu [37], tak TurboCrypt je nástroj pro šifrování pevných disků a celých oddílů, právě jako nástroje TrueCrypt nebo VeraCrypt. Umožňuje také vytvoření tzv. „hidden“ oddílů a umožňuje volbu šifrovacího algoritmu → volba šifry AES nebo polymorfní blokové šifry s délkou bloků 1024 bitů. Doporučené je volit polymorfní šifru. Princip polymorfní šifry není dohledatelný, ale je možné si udělat představu z popisu další verze, která má svůj „white-paper“ [38] nebo z této webové stránky [39].

V těchto zdrojích, které vznikly do roku 2010 a které popisují a vyzdvihují základní myšlenku polymorfních šifer včetně ilustrace výhod (viz tabulka č. 1 a 2) se lze dočíst následující.

- Základní myšlenkou je využití šifrování pomocí kombinace šifer → v základu i dostupných algoritmů jako byly MARS, Twofish, RC6 nebo AES. Kaskádové šifrování jako v systémech TrueCrypt a VeraCrypt.
- Využití variabilní délky bloků, kdy délka bloků by měla být ideálně stejně dlouhá, jako je délka otevřených dat → otevřená data ve formě jednoho bloku dat pro šifrování. → Eliminace nutnosti využít padding.
- Šifrování bloku je rozděleno ve variabilně dlouhé kusy, kdy i klíč je rozdělen na stejné bloky. Jednotlivé bloky jsou pak šifrované samostatně.
- Výpočet klíče na začátku, který bude dlouhý dle potřeby délky otevřených dat a který bude ve formě pseudo-náhodného řetězce bitů. → Využití dynamicky volené kombinace jednoduchých PRNG.
- Pojmenování šifry → Giant Block Size Polymorphic Cipher
- Součásti je návrh polymorfního PRNG a polymorfní hashovací funkce.
- Konkrétní implementaci nelze dohledat a nelze stáhnout.
- Copyright vlastněn firmou PMC Ciphers, Inc. & Global IP Telecommunications, Ltd. (Německo).

Tabulka 1 – Souhrn výhod šifry → Polymorphic Giant Block Size Cipher – první část [39]

Design goal	Polymorphic Giant Block Size Cipher	Conventional Ciphers
Large and variable block size	**Block size is only limited by the resources of the target computer(s). Target systems should run at 500MHz or higher and more than 10Mbyte free RAM should be available. The Strict Avalanche Criterion is thus met perfectly.**	**Not supported at all. Ciphers like AES need little more than 1Kbyte of machine code and a microcontroller typically used in cheap smart cards and washing machines (approx. 20.000 transistors) to run. It is conceivable that such conventional ciphers could have been hardened against all kinds of attacks if more complex implementtations would have been the target.**
No padding to reach block granularity shall be necessary	**Block size is totally variable and blocks keep their length => no padding required, which results in no information being transmitted in vein.**	**DES: 8 byte block granularity, AES: 16 byte block granularity ⇨ Padding required A 2048 bit conventional block cipher would require padding to 256 byte blocks resulting in dramatic increase in data traffic if used for the encryption of TCP or UDP data packets.**
Partitioning of extremely big blocks at arbitrary position	**Blocks that are too big to handle are truncated into sub-blocks with block sizes that are determined by the key as well as the length of the original block.**	**Not supported at all. AES, DES and all other well-known block ciphers feature fixed block sizes.**
Resistance against all known attacks	**Due to its variable nature are Polymorphic Ciphers not susceptible to typical attacks that target specific characteristics and/or known weaknesses of fixed ciphers. Brute Force is although applicable to any cipher.**	**AES can be broken easily by DPA (Differential Power Attack) on small microprocessors and microcontrollers.**
Resistance to future attacks that may cut effective key size by ½ or even 2/3	**Cutting of effective key size by ¾ would result in still extremely high complexity of $O(2^{256})$ or higher, which is regarded as totally safe for the next trillion years.**	**Cutting of effective key size by ½ results in an extremely low complexity of 2^{64}. The cipher would be regarded as being broken.**
Extremely long key setup time	**> 100ms on a modern microprocessor make comparably short keys safe against Brute Force attacks conducted on a few machines. Extremely long key setup time increases energy consumption multiplied by the time needed for Brute Force by factor 2.000.000.**	**<1µs help attackers to try each and every password combination. This is highly dangerous if short passwords are being used to protect data.**

Tabulka 2 - Souhrn výhod šifry → Polymorphic Giant Block Size Cipher – druhá část [39]

Platform independence	Runs on any 32 or 64 bit microprocessor or microcontroller	Runs on any 8-, 16-, 32- and 64 bit microprocessor and microcontroller
Polymorphism and data dependent selection of functions	**The cipher is not only completely variable, but also is the block size huge and unpredictable if truncation is performed. No static weakness is exhibited.**	**Classic ciphers are static and can thus be thoroughly reverse-engineered and analyzed. Cryptanalysis of a mechanism that does always exactly the same is somewhat easier than for a mechanism that never executes the same operation twice.**
Use of large amounts of re-sources	**1 Mbit internal state requires at least approx. 8 million transistor equivalents to run. This alone makes Brute Force Attack more difficult and much more expensive compared with conventional ciphers.**	Less than 50.000 transistor functions are required to build an AES block. **Approx. 1.000.000 AES blocks can run in parallel on an 8" wafer to try and break a code using Brute Force.**
Attacks need to be expensive for an attacker	**The proposed cipher requires a lot of resources and extremely much time for key setup, an attacker requires a "time x resources product" of approx. 200.000 times compared with AES Rijndael when using keys with a similar length.**	**Trying different AES keys requires 50.000 transistor equivalents and less than 1µs. This isn't really all that much. This is a REAL weakness.**
High speed	**1500 Mbit/s on an Intel Core i7 950 (3.06GHz) (64 bit C++ code, 1024 byte block length)**	1000 Mbit/s on an Intel Core Core i7 950 (3.06GHz) (64 bit C++ code)
Proven security	**Three round Luby Rackoff features proven security (the mathematical proof is contained in the PDF doc that describes the cipher); polymorphic encryption is increasingly popular among experts but it's probably impossible to prove security of the entire cipher.**	**Security is not proven. Extensive peer review indicates that the cipher could be broken in the future:** **For 128-bit Rijndael, the problem of recovering the secret key from one single plaintext can be written as a system of 8000 quadratic equations with 1600 binary unknowns.** **Recently has a new related-key boomerang attack on the full AES-192 and the full AES-256 been found by Biryukov and Khovratovich. A 256 bit key is reduced to a 119 bit key when using AES-256. The attack is not applicable to 128 bit keys.**

Z výše uvedeného lze získat obraz o funkcionalitě a základní myšlence polymorfního šifrování. Zároveň, jak je správně uvedeno, tak pravděpodobným důvodem, že polymorfní šifry nejsou v praxi široce využívány je problematika jejich standardizace. Protože u komplexních polymorfních systémů záleží na aktuálních okolnostech a vazby v systému jsou tak složité, že jejich analýza je natolik komplexní, že s její pomocí nelze potvrdit, ale ani vyvrátit jejich bezpečnost.

Stránka ciphers.de je v porovnání s „white-paper" dokumentem [38] relativně nová → copyright roku 2018, kdy firma nabízí kryptografická řešení. Pokud bychom chtěli porovnat aktuální stav se stavem z roku 2010, tak se lze

domnívat, že systém prošel úpravami a změnami. Nicméně se bude jednat o neveřejný uzavřený systém.

4.4.2 Kostadin Bajalcaliev Cryptography Page

Jak bylo zmíněno v úvodu kapitoly, tak dalším velmi zajímavým zdrojem je webová stránka Kostadina Bajalcalieva z Makedonie, který výzkumu polymorfního šifrování věnoval roky 1995 až 2000. Během těchto let jsou zde zachyceny dokumentace a zdrojové kódy následujících algoritmů:

- Anigma – polymorfní boková šifra s jádrem reverzibilně upraveného autorova hashovacího algoritmu MEX
- SQ1 – polymorfní proudová šifra
- SQ2, SQ3 (Z876) – optimalizace a vylepšení implementace šifry Anigma (podrobnosti bohužel není možné dohledat)
- MEX – polymorfní hashovací funkce

Na stránce je možné dohledat závěrečnou maturitní práci autora, která byla vypracovaná jako „projekt“ SQ a optimalizace implementace šifry Anigma a formální vymezení myšlenky šifry Anigma. Hlavním výstupem a závěrečnou prací autora je dokument z roku 2001, 7. května, který má název „Quasi Functions / Polymorphic Encryption“. Tento dokument shrnuje veškerý vývoj autora do přibližně 10 stran textu.[1] Materiál sumarizuje výzkum do roku 2001 a popisuje problematiku včetně matematických vyjádření. Jsou zde uvedeny varianty šifer SQ5 a SQ6. SQ5 je uvedena jako nejnovější varianta SQ3 a SQ6 je proudová šifra s implementací navržených kvazi funkcí. Podrobnější informace či zdrojový kód už není na stránkách dostupný → práce odkazuje na neexistující stránku.

4.5 Režimy činnosti využívané v praxi

Problematika režimů činnosti není tak obsáhlá, jako problematika kryptografie blokových šifer. Dlouhou dobu se využívaly klasické režimy činnosti standardizované NIST [13]; [14], ale v široké povědomí se dostávají režimy činnosti (zejména s problematikou přenosu dat internetem) a začínají využívat režimy činnosti rozšířené o autentizaci pomocí připojených dat → souhrnně „Authenticated Encryption with Associated Data“ (zkráceně AEAD). Proto

[1] *Dokument je dostupný zde - https://kbajalc.tripod.com/algo/pme/polym.html*

můžeme rozdělit problematiku na klasické režimy činnosti a moderní režimy činnosti, kdy ty druhé využívají principu AEAD. [41]

4.5.1 Klasické režimy činnosti

Klasické režimy činnosti a zároveň standardizované režimy činností dle NIST [13]; [14]:

- Cipher block chaining (CBC)
- Output Feedback (OFB)
- Cipher Feedback (CFB)
- Counter (CTR)
- Electronic Codebook (ECB, viz obrázek číslo 2) → nezajišťuje změnu dat a difúzi

Každý z nich provádí předem danou manipulaci s daty. CBC před šifrováním následujícího bloku dat sečte pomocí funkce exkluzivního součtu → „exclusive or“ (XOR) aktuálně šifrovaný blok P_i s předchozím zašifrovaným blokem C_{i-1}. Režim činnosti OFB využívá předchozí zašifrovaný blok C_{i-1} jako vstup P_i a až po zašifrování použije původní P_i a sečte jej pomocí operace XOR s aktuálním blokem C_i. V tomto případě je nutné jako první šifrovaný blok P1 použít tzv. inicializační vektor (*IV*). Inicializační vektor je náhodně generovaný blok dat o délce bloku dat.

V dnešní době se problematika využití režimů činnosti blokových šifer mírně změnila. Zejména zmíněný režim činnosti CBC už není doporučen k využití, protože obsahuje bezpečnostní rizika a problémy. Jedná se zejména o útoky [47]:

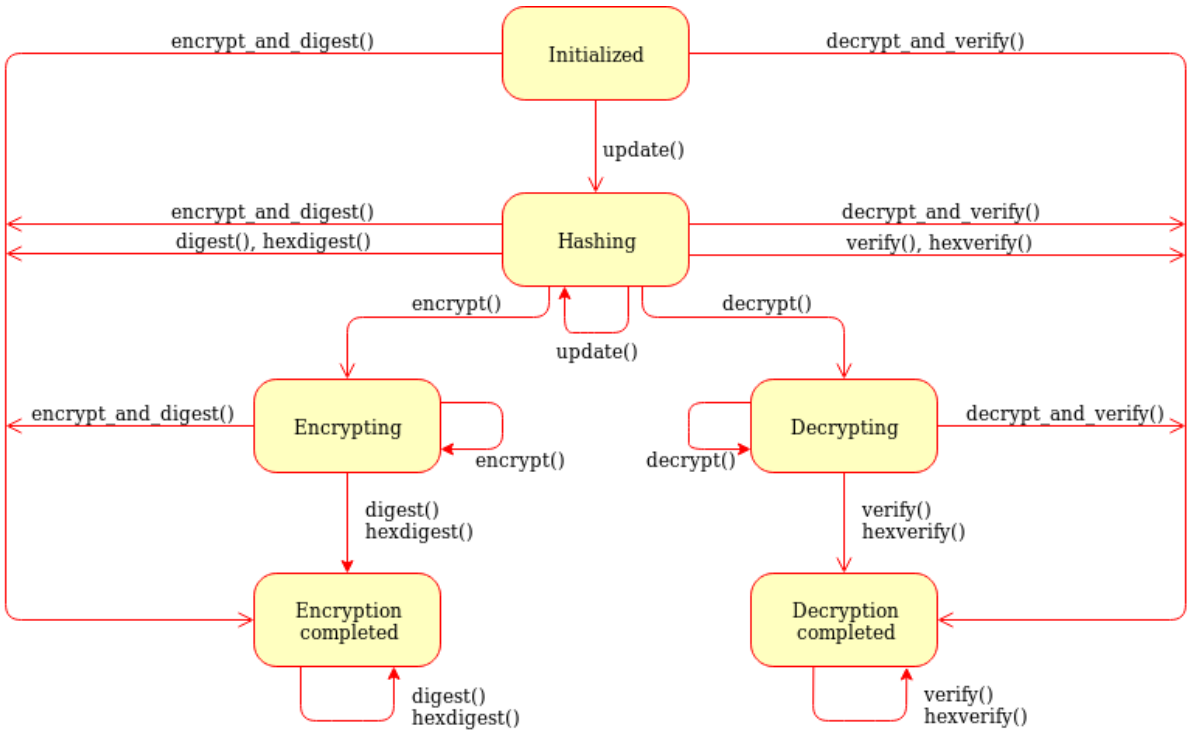
- BEAST
- POODLE
- CRIME

4.5.2 Moderní režimy činnosti - AEAD

Doporučené režimy činnosti lze najít v souvislosti s využitím nejnovější verze kryptografického protokolu pro šifrování internetové komunikace TLS 1.3

[48][2]. Kdy do verze 1.2 lze využít režim činnosti CBC, ale od verze 1.3 je považováno za bezpečné využití pouze režimů GCM a CCM, které spadají do speciální části režimů činnosti blokových šifer; do tzv. moderních režimů činnosti blokových šifer s principem šifrování - AEAD.[41] Výsledkem po zašifrování pomocí této skupiny režimů není jen šifrový text, ale i tzv. autentizační značka (authentication tag). Dalším využívaným režimem činnosti je například XTS (XEX), který principiálně spadá spíše do klasických režimů činnosti.[50] Využívaný v rámci šifrování nástrojem Veracrypt.

Princip fungování moderních režimů činnosti s možností autentizace lze vidět na následujícím schématu.



Obrázek 3 - Obecné schéma fungování režimů činnosti v rámci AEAD [41]

Mezi zástupce moderních režimů činnosti lze zařadit následující:

- CCM [42]
- EAX [43]

---

[2] *zdroj wikipedia, zde je potřeba brát v potaz možnou irelevanci informací, ale tento zdroj je uveden jako souhrnný.*

- GCM [44]
- SIV [45]
- OCB [46]

Jak je uvedeno v kapitole 7.2.3, tak by bylo v rámci budoucího rozšíření TSS teoretické možné využití soli pro „zasolení“ jako autentizační tag a tudíž praktickou aplikaci principu AEAD.

4.5.3 Inicializační vektor a nonce

Některé režimy činnosti souvisejí s nutností vygenerování tzv. inicializačního vektoru nebo *nonce*. Proces lze přirovnat k zavedení náhodnosti do systému, aby byly na začátku odstranění markanty otevřeného textu. *Nonce* – v překladu šum je náhodná startovací sekvenci dat, která je zejména zvolena pro režimy odvozené od režimu CTR – CCM, EAX a další. Režim činnosti GCM využívá *IV*, který využívá i režim CBC. Každopádně, pokud se jedná o *IV* nebo *nonce* tak platí, že obě dvě hodnoty by měly být vygenerované náhodně a tyto hodnoty je nutné přenášet spolu se šifrovým textem. Pro úspěšné dešifrování zašifrovaných dat je tedy potřeba tyto hodnoty znát.

Systém solení v rámci TSS nutnost tyto hodnoty eliminuje, návrh je podrobně rozveden v kapitole 7.2.

4.6 Shrnutí současného stavu řešené problematiky

Na základě výše popsaného lze současný stav řešené problematiky shrnout následovně.

Existuje celá řada algoritmů v oblasti kryptografie blokových šifer, která je prakticky využívána. S tím existují režimy činnosti, algoritmy pro odvození klíčů a nespočet praktických implementací v kryptografické nástroje, systémy a protokoly. Nicméně drtivá většina z nich má fixní → časově či parametrově nezávislé chování, strukturu a průběh.

Existují i zástupci algoritmů, které lze alespoň částečně zařadit mezi polymorfní. Každopádně se jedná pouze o částečnou polymorfizaci.

Dále lze narazit i na komplexněji polymorfní systémy či teorie, které ovšem nenašli praktického uplatnění či aplikace.

Rozhodně lze říci, že problematika polymorfních struktur, nejen v oblasti symetrické kryptografie, by zasluhovala hlubší pozornost a výzkum. Lze se jen domnívat, proč tomu tak není, kdy jeden z možných důvodů je velmi obtížná (až nereálná) analýza těchto systémů/algoritmů. Což znemožňuje potvrzení bezpečnosti či standardizaci.

V porovnání se systémem TSS, který je navržen v disertační práci, tak nebylo možné dohledat systém, který by principy polymorfnosti využíval v takové míře. Systém, který ve svém komplexním návrhu kombinuje všechny prostředky nutné pro postavení kryptografického systému včetně šifrovacího jádra → polymorfní blokové šifry.

5 ZÁKLADNÍ MYŠLENKA VYBUDOVÁNÍ POLYMORFNÍHO ŠIFROVACÍHO SYSTÉMU

V rámci této kapitoly je rozebrána základní myšlenka pro vybudování polymorfního šifrovacího systému. Rozebírání možnosti a způsoby, jakými by bylo možné transformovat blokové šifry v polymorfní. V rámci kapitoly jsou vybrány jednotlivé části blokových šifer, které budou následně upraveny a změněny na polymorfní struktury.

Jak už bylo uvedeno výše, v rámci moderní kryptologie se využívají nebo byly navrženy algoritmy, které jsou výhradně navrženy tak, že celý průběh šifrování je znám a jediná tajná informace je šifrovací klíč. Tento fakt vychází ze základní myšlenky, že algoritmus musí být veřejný. Bez pochyby se jedná o validní a velmi důležitý faktor, který je potřebné dodržet, jak se ukázalo v minulosti. Jedná se o problematiku „Security through obscurity“, problematika rozebrána výše, v rámci kapitoly 2.4.1. Odborná komunita se výhradně shodne, že šifrovací systém by měl být založen na matematických principem s ohledem na otevřenost systému. Neměl by spoléhat pouze na utajení algoritmu šifrovacího systému. Nicméně, pokud se systém navržen na matematických principech, lze v tomto případě ještě doplnit bezpečnost o neveřejnost algoritmu. To ovšem za předpokladu, že je algoritmus dostatečně bezpečný a byl podroben kvalitní kryptoanalýze. V opačném případě je jeho veřejnost výhodou z důvodu, že komunita, uživatelé a kryptoanalytici mají možnost provést testování těchto šifrovacích systémů. Protože jakákoliv chyba může být odhalena kýmkoliv a vzápětí odstraněna/opravena. Celkově nelze nikdy vyloučit absenci chyb v rámci systému, ale můžeme tvrdit, že pokud byl systém podroben více testů, bez nalezeného bezpečnostního problému, tak jej lze považovat za bezpečnější.

Z výše uvedených podmínek je právě šifra AES natolik oblíbená a i využívaná, protože bylo provedeno nespočet pokusů o prolomení. Nespočet vědeckých prací se zabýval její kryptoanalýzu a doposud nebyl nalezen žádný efektivní způsob na prolomení či snížení bezpečnosti této šifry. Lze uvést i příklad z jiné oblasti – zobecněná teorie relativity je čím dál více potvrzována pokusy vědců a jejich snahami o její vyvrácení. Tím můžeme říct, že jakýkoliv negativní pokus o prolomení šifry je zároveň vhodný prostředek pro potvrzení bezpečnosti.

Výhoda otevřenosti šifrovacího systému je nezpochybnitelná a v rámci této disertační práce je navržen kryptografický systém, který je v souladu s tímto pravidlem. Nicméně, tato disertační práce se snaží tuto problematiku prohloubit a rozšířit o následující možnost; konkrétně o návrh systému, který by ve své podstatě byl otevřený, veřejný a postaven na matematických principech. Zároveň aby jeho šifrovací proces byl tajný stejně jako využití tajného šifrovacího klíče.

Navržený systém tedy je vybudován na známých operacích a má veřejnou podobu, ale konkrétní nastavení a vlastnosti se budou měnit spolu se vstupními podmínkami. Můžeme tedy mluvit o tom, že celkový proces šifrování, včetně parametrů a vlastností šifrovacího systému, se polymorfně mění v závislosti na změnách počátečních/aktuálních hodnot v rámci systému. Výsledkem je šifrovací systém, který kombinuje výhody otevřenosti systému, ale zároveň jeho konkrétní průběh je proměnlivý. Bez nadsázky můžeme říci, že vytvořený kryptografický systém v sobě zahrnuje složku t – čas. Neboli při každém šifrování se průběh a jednotlivé parametry mění v závislosti na čase, i když nedojde ke změně vstupních dat/otevřeného textu nebo klíče/klíčového souboru.

V rámci disertační práce byl vybudován kompletní systém, který je na základě uživatelských vstupních parametrů nastaven a inicializován pro šifrování prvního bloku dat a následně je před šifrováním dalšího bloku dat na základě aktuální parametrů modifikován a upraven. Návrh je proveden tak, aby šifrování aktuálního bloku dat bylo neznámé, a to bez znalosti vstupních parametrů a klíče. Celkově, aby před samotným šifrováním nebylo známé, za jakých podmínek a parametrů je šifrování prováděno. Proces a parametry jsou tak v maximální míře závislé na tajném klíči, ale i na dalších parametrech, jako jsou náhodná vstupní data či otevřená data.

Pro navržený kryptografický systém bylo vymyšlené kódové označení TSS, kdy nejnovější verze nese název s adaptací – „Triply Salted Smallie“ („Trojitě nasolené Smallie). V rámci disertační práce je rozepsána hlavně verze „Triply Salted Smallie“. Jedná se o variantu, kdy v rámci solení bylo využito tří samostatně generovaných solí.

5.1 Návrh systému a aplikovatelnost

Celkový návrh systému byl proveden v souladu s výše uvedenou filozofií. Aby došlo k co nejvyšší míře polymorfnosti, závislosti podmínek a stavů na prvotních tajných datech (klíči), vstupních parametrech a nastavení a neposlední řadě na náhodně generovaných datech (soli). Dále byl celý systém navržen tak, že jednotlivé části byly přizpůsobeny principu polymorfnosti, tudíž většina funkci a částí systému vytvořena za účelem vybudování unikátního a uceleného kryptografického systému. To znamená, že případná aplikace a použití jednotlivých složek je možná a aplikovatelná na jiné algoritmy (blokové šifry), ale byla by zde nutná korekce systému. Nejjednodušší možnosti přizpůsobení vůči aktuálním systému by bylo vhodné nahrazení šifrovacího jádra jiným. Například využití šifry AES a zachování key-managementu a systému solení. To ovšem za výběru parametrů, které by bylo možné využít v kombinaci se šifrou AES. Problematika je rozebrána podrobněji v části 7.4.

Před samotným návrhem bylo potřeba zvolit parametry a vlastnosti, které budou mít vliv na průběh šifrování. Tímto se zabývá následující podkapitola.

5.2 Volba vlastností a parametrů pro polymorfnost šifrování

Šifrovací systém je závislý na následujících parametrech a celkový proces lze rozdělit následovně; příprava parametrů a vlastnosti pro zašifrování prvního bloku dat, příprava a manipulace s algoritmem pro šifrování následujícího bloku dat. Jako první je potřeba vytvořit šifrovací klíč pro šifrování prvního bloku dat. Proces vytvoření klíče bude ovlivněn následujícími parametry a bude se měnit na základě vstupních dat, viz kapitola o přípravě šifrovacího klíče 7.3:

- Tajná data libovolné velikosti – klíčový soubor
- Náhodná data – sůl (soli) – rozebráno v kapitole 7.2

Proces přípravy vlastností a parametrů pro šifrování prvního bloku dat, inicializace šifrovacího systému, bude ovlivněn na základě:

- Tajného klíče (inicializačního vektoru)
- Tajných vstupních parametrů nastavených uživatelem
- Defaultní substituční tabulky

Šifrování a parametry druhého a následujících bloků budou ovlivněny na základě:

- Klíče použitého pro šifrování předchozího bloku
- Otevřených dat předchozího bloku
- Posledních zašifrovaných dat
- Předchozích parametrů

Níže je vypsán výčet vlastností a parametrů, které se budou dynamicky měnit v rámci šifrovacího procesu. Jedná se o:

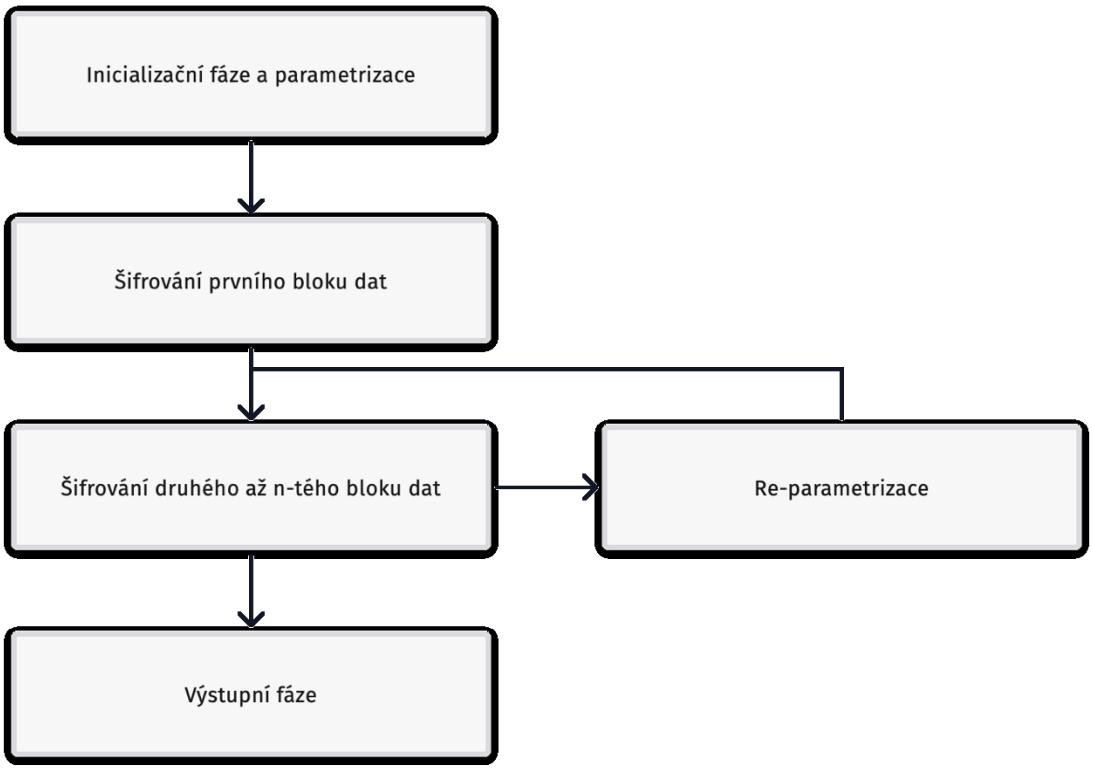
- Klíč a způsob jeho odvození
- Velikost šifrovaného bloku
- Parametry solení
- Počet rund
- Pořadí šifrovacích operací
- Parametry pro parametrizaci, včetně řídících
- Průběh šifrovacích operací
- Parametry a průběh režimu činnosti – klíčovací management
- Substituční tabulka

Parametrizace, proces a operace spojené s polymorfností jsou rozebrány níže v kapitole 7.4. Nicméně, v budoucnu by bylo možné přidat nespočet dalších parametrů a vlastností, které by bylo možné upravit, aby se dynamicky měnili v průběhu šifrování. Obdobně je možné manipulovat s aktuálními částmi a upravovat je. Například pro potřeby rychlosti šifrování, pro potřeby paralelizace či vůči konkrétní aplikaci.

6 VYTVOŘENÝ KRYPTOGRAFICKÝ SYSTÉM - TSS

Tato kapitola pojednává o vytvořeném polymorfním kryptografickém systému. Systém byl navržen v souladu s popisem výše. Tvorba systému probíhala v několika fázích a systém byl několikrát upraven a vylepšen. Kapitola popisuje a ukazuje finální návrh. Systém je i částečně postaven na základě šifrovacího jádra diplomové práce autora [11], kdy bylo zmíněné šifrovací jádro upraveno a vylepšeno. Jednotlivým částem systému a jejich popisu se věnuje text v podkapitolách.

V základu jde výsledný polymorfně založený šifrovací systém znázornit následujícím diagramem.



Obrázek 4 - Schéma polymorfního kryptografického systému – TSS

Na základě výše zobrazeného schématu lze průběh šifrování pomocí výsledného šifrovacího systému rozdělit na pět základních částí. Jednotlivé části budou blíže popsané v následujících kapitolách.

Jedná se o části:

1. Inicializace a první parametrizace.
2. Šifrování prvního bloku dat.
3. Re-parametrizace – příprava klíče a parametrů pro šifrování dalšího bloku.
4. Šifrování dalších bloků otevřeného textu.
5. Výstupní fáze – tvorba finálního šifrového textu.

Je důležité poznamenat, že šifrování prvního nebo jakéhokoliv dalšího bloku probíhá stejně (aplikování šifrovacího jádra/nosné šifrovací funkce), ale každý blok je šifrován/dešifrován za pomocí jiného klíče a na základě jiných parametrů a dochází k adaptaci šifrovacího jádra na základě předchozích podmínek/parametrů. Rozdíl je v prvotní parametrizaci a pak následné re-parametrizaci mezi šifrováním dalších bloků.

V rámci dešifrování se postupuje prakticky stejně, akorát rozdíl je v inicializační fázi a parametrizaci a dále ve výstupní fázi. Hlavní rozdílem je manipulace se solemi a úprava otevřeného/šifrového textu. V případě šifrování se soli používají na „předsolení“ a „zasolení“ – připojení k otevřenému textu zepředu a zezadu.

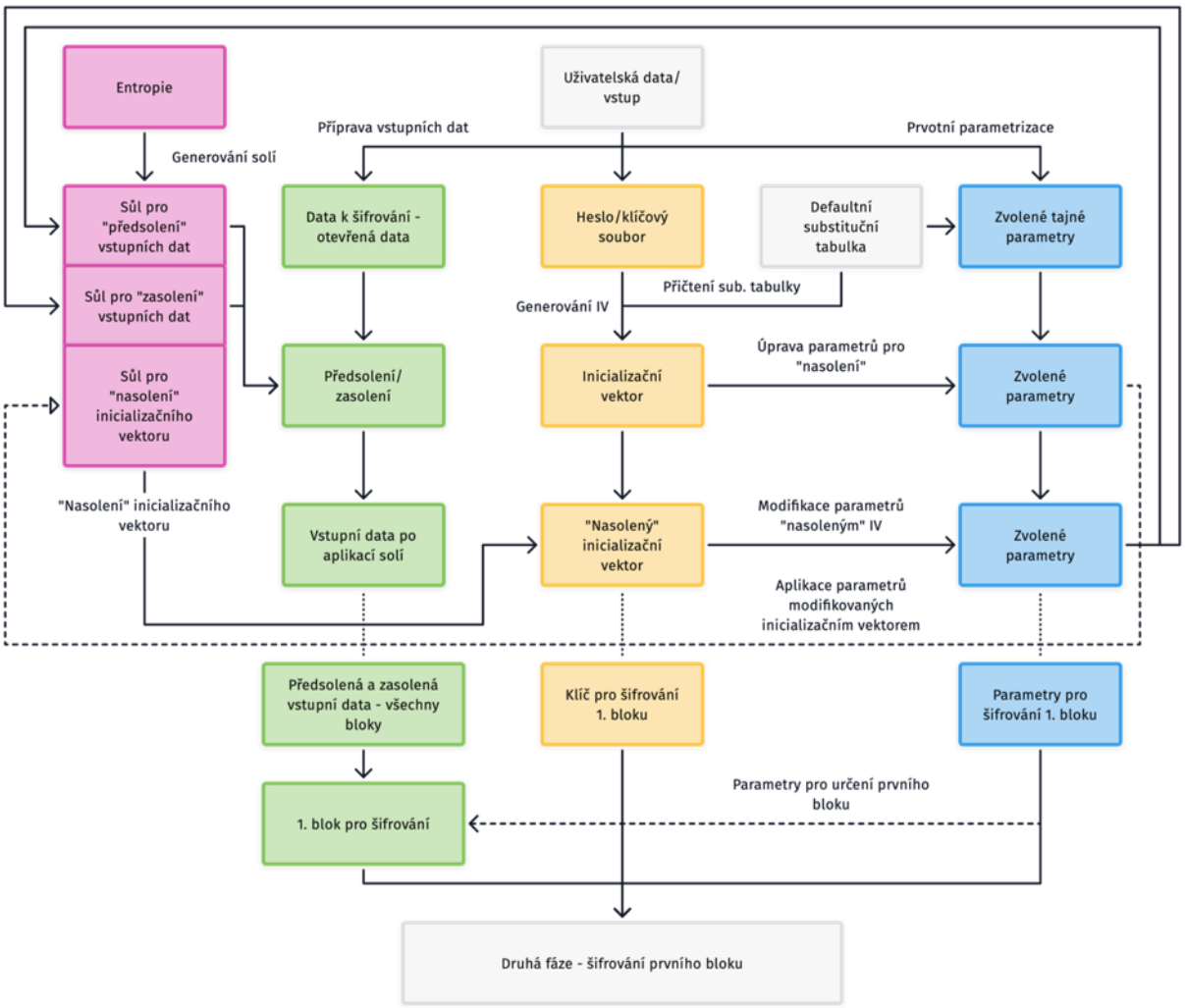
Tyto soli jsou dále šifrovány, takže v rámci dešifrování je možné „odsolení“ – odstranění soli zepředu a zezadu až po dešifrování všech dat. Obdobně je to se solí, která slouží k „nasolení“ inicializačního vektoru. Při šifrování dochází v závěrečné fázi k vmísení soli pro „nasolení“ to výsledného šifrového textu na základě pravidel inicializačního vektoru a parametrů. Při dešifrování je ale nutné ještě před započetím dešifrování patřičnou sůl pro „nasolení“ získat ze šifrového textu, což probíhá opět na základě inicializačního vektoru a parametrů.

6.1 První část – inicializace fáze a úvodní parametrizace

V rámci první fáze dochází k následujícím krokům:

1. Získání vstupních dat od uživatele – otevřená data
2. Získání parametrů od uživatele – tajné nastavení vstupních parametrů
3. Získání dat pro potřeby vygenerování prvního šifrovacího klíče – inicializačního vektoru
4. Sběr entropie a generování náhodných dat – modifikace parametrů a inicializačního vektoru

Proces lze znázornit pomocí následujícího diagramu.



Obrázek 5 - Schéma fáze první - příprava na šifrování prvního bloku dat

Výsledkem je zpracování uživatelského vstupu pro určení:

- Prvního blok data pro šifrování
- Klíče pro šifrování prvního bloku dat
- Parametrů pro šifrování prvního bloku dat (zahrnující substituční tabulku)

Jednotlivé fáze a části jsou pro přehlednost barevně odlišeny:

- Růžová barva reprezentuje sběr entropie a přípravu solí pro „předsolení“, „zasolení“ a „nasolení“ vstupních dat, konkrétní postup „předsolení“, „zasolení“ a „nasolení“ je rozveden v kapitole 7.4.
- Zelená barva reprezentuje postup manipulace se vstupními daty až po odvození prvního bloku dat pro šifrování.
- Žlutá barva reprezentuje proces a kroky pro určení klíče pro šifrování první bloku dat.

- Modrá barva reprezentuje proces a průběh určení parametrů, které jsou použité pro šifrování prvního bloku.

6.2 Dešifrování

Při dešifrování se postupuje v rámci první fáze obdobně. Odvození klíče pro dešifrování prvního bloku je naprosto stejné. Jako první jsou načtena uživatelská data pro jeho odvození (heslo/klíčový soubor) a dále tajné parametry. Vypočte se inicializační vektor, který slouží pro získání soli k „nasolení“, který byl při šifrování začleněn do šifrových dat. Tento inicializační vektor je použitý k výpočtu parametrů pozice soli k „nasolení“. Jakmile je známá pozice soli, lze tuto sůl extrahovat a „nasolit“ inicializační vektor. Ten je následně využitý k výpočtu parametrů pro dešifrování prvního bloku a výpočtu parametrů solí k „předsolení“ a „zasolení“, které bude nutné v rámci poslední fáze z dešifrovaných dat odstranit. V rámci dešifrování dat se postupuje stejně, pouze s rozdílem aplikace inverzních funkcí v rámci šifrovacího jádra. Odvození klíčů pro další bloky dat je naprosto stejné jako při procesu šifrování.

6.3 Souhrn vlastnosti kryptografického systému TSS

Výsledné vlastnosti vytvořeného polymorfního kryptografického systému Smallie, konkrétně varianty Triply Salted Smallie lze shrnout následovně.

6.3.1 Velikost klíče pro šifrování

V základu je délka klíče nastavena na 111 bajtů. Výběr 111 bajtů byl zvolen nejenom kvůli velikosti klíčového prostoru následovně.

$$Velikost\ klíčového\ prostoru = 2^{888} \cong 2{,}06 * 10^{267}$$

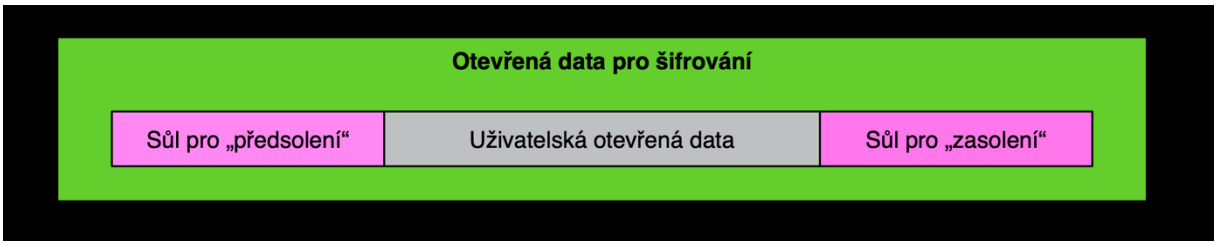
Velikost klíče byla zvolena nejen kvůli odolnosti proti útoku hrubou silou, ale dále i z důvodu nutnosti zvýšení počtu kombinací pro potřeby začlenění dalších závislostí. Zejména se jedná o potřebu generování a manipulaci s parametry celého systému, viz kapitola věnovaná parametrizaci.

6.3.2 Otevřená data

Otevřená data; neboli data určené k šifrování, které lze šifrovat nejsou prakticky nijak omezená a systém je navržen tak, aby bylo možné šifrovat data velikost 0 B až po libovolně velká data. Celý systém je díky variabilní délce bloků schopen šifrovat bloky libovolné délky, a to bez nutnosti data doplňovat (problematický padding). Vstupní data jsou doplněna o náhodné znaky zezadu – „zasoleny“, aby došlo k zastření informací o velikosti vstupních dat. Obdobně jsou uživatelská otevřená data doplněna o data ve formě soli zepředu, tzv. „předsolení“. Má to za důsledek vnesení entropie do otevřeného textu a zároveň

vnesení entropie do tvorby šifrovacího procesu následujících bloků dat, proto se jeví celé šifrování včetně všech vlastností jako náhodné.

Sůl, která slouží k „předsolení“, tak i sůl, která slouží k „zasolení“ jsou spolu s uživatelskými otevřenými daty zašifrovány a zahrnuty do výsledných šifrových dat. Výsledná otevřená data po „předsolení“ a „zasolení“ popisuje následující diagram, kdy barvy jsou významovou reprezentací z fáze 1, viz obrázek číslo 6.



Obrázek 6 - Otevřená data po "předsolení" a "zasolení"

Z obrázku výše jde tedy vidět, že délka dat pro šifrování je delší než samotná data od uživatele. Délka výsledných šifrových dat bude rozebrána v následující podkapitole číslo 6.3.3. Délka je ovlivněna také délkou soli sloužící pro „nasolení“ šifrovacího klíče z inicializačního vektoru. Délku otevřených dat si označme l_o. Délku soli pro „předsolení“ si označme l_p a pro označení délky soli pro „zasolení“ si zvolíme l_z. Původní uživatelská otevřená data budou reprezentované pomocí l_{op}, Potom lze výslednou délku otevřených dat l_o vyjádřit následovně.

$$l_o = l_p + l_{op} + l_z$$

Kdy délka l_p není ničím limitována a délky l_p a l_z jsou určeny v rámci první fáze pomocí uživatelského vstupního nastavení a na základě tajného inicializačního vektoru, který byl „nasolen“ pomocí soli, tudíž jsou délky závislé na prvním šifrovacím klíči. Délky l_p a l_z jsou voleny na základě rozdílných parametrů tak, aby byly na sobě nezávislé.

Celkově lze říci, že délku l_o ovlivňují následující faktory a parametry:

- Vstupní parametry pro určení délky „předsolení“
- Vstupní parametry pro určení délky „zasolení“
- Parametry pro určení délky „předsolení“ modifikovány na základě „nasoleného“ inicializačního vektoru
- Parametry pro určení délky „zasolení“ modifikovány na základě „nasoleného“ inicializačního vektoru

Inicializační vektor dále určují vstupní parametry od uživatele a tajná data, která jsou označená jako klíčová data (heslo či klíčový soubor). Následně do procesu vstupují náhodně generovaná data ve formě soli k „nasolení“ inicializačního vektoru. Celkově tedy „nasolený“ inicializační vektor slouží k převedení vstupních tajných parametrů od uživatele ve výsledné parametry sloužící k výpočtu finálních délek solí pro „předsolení“ a „zasolení“.

Tudíž můžeme tvrdit, že délku solí neurčují jenom tajné uživatelské parametry, ale i náhodná data, a proto délka otevřeného textu se chová náhodně. Délku otevřených dat, jakožto i výsledných šifrových dat není možné před samotným šifrováním určit, díky entropii náhodné soli a tajné volbě nastavení uživatelem. Délky solí pro „předsolení“ a „zasolení“ byly zvoleny následovně:

- Délky solí l_p a l_z jsou minimálně 111 bajtů + 0 až 255 bajtů v závislosti na parametrech a „nasoleném“ $IV \rightarrow$ hodnoty celých čísel na intervalu $\langle 111; 366 \rangle$

Základní délka 111 bajtů je zvolena z důvodu, že je to základní velikost klíče a velikost šifrovaného bloku dat. Variabilní část soli, kterou můžeme označit l_{vp} a l_{vz} byla zvolena jako hodnota celého čísla na intervalu $\langle 0; 255 \rangle$, protože je to hodnota určena na parametricky zvoleném jednom bajtu „nasoleného“ *IV*. Podrobněji je určení délek l_p a l_z rozebráno v kapitole 7.4.4. Vyjádření rovnicemi délek je následující.

$$l_p = 111 + \ l_{vp}$$

$$l_z = 111 + \ l_{vz}$$

Celková délka otevřených dat před šifrováním je hodnota náhodně se pohybující celé číslo na intervalu $\langle l_{op} + 222; l_{op} + 732 \rangle$, kdy délka bude minimálně na intervalu $\langle 222; 732 \rangle$ a to za předpokladu, že dojde k šifrování prázdných otevřených dat.

6.3.3 Šifrová data

Výsledná šifrová data určuje spousta faktorů. V této podkapitole si je uvedeme a také podrobněji rozebereme výslednou délku šifrových dat. Výsledná šifrová data a jejich tvar je dán hlavně nejenom vstupními otevřenými daty, která jsou ovlivněna, jak bylo uvedeno výše procesem „předsolení“ a „zasolení“, ale zejména šifrovacím procesem, který je silně parametrizován.

V závislosti na parametrizaci se mění šifrovací proces, tudíž i stejná otevřená uživatelská data mohou být šifrovány jiným způsobem a za jiných parametrů. V neposlední řadě je v rámci „nasolení“ do šifrového textu vložena

sůl, která byla použita pro „nasolení“ *IV*. Na základě popsaného, můžeme říci, že obsah šifrových dat bude ovlivněn šifrovacím procesem a s tím souvisejícími nastaveními a parametry a dále procesem „nasolení“, což ovlivní nejen obsah, ale i samotnou délku šifrových dat. Nyní si to rozeberme podrobněji.

Délka šifrových dat označme $l_š$ a její hodnota je vymezena na základě délky zašifrovaných otevřených dat, viz podkapitola výše. Tedy hodnotou l_o a dále délkou soli, která byla použita na „nasolení“ *IV*, označme l_n. Délku $l_š$ lze vyjádřit následovnou rovnicí.

$$l_{\check{s}} = l_o + l_n$$

Jak bylo uvedeno v rámci podkapitoly 6.3.2, tak můžeme říci, že hodnota l_o je náhodná, protože je ovlivněna „nasoleným“ *IV*. Co se týká délky l_n, tak ta je ovlivněná na základě následujících faktorů:

- Vstupní tajné parametry zvolené uživatelem pro „nasolení“
- Tajných klíčových dat -> následně *IV* (přepočet vstupních parametrů), proces je podrobně rozebrán v kapitole 7.2 – proces solení
- Defaultní substituční tabulkou

Tudíž délka soli pro „nasolení“ v tomto případě není náhodná, ale odvozena vždy stejně na základě vstupních parametrů a tajných klíčových dat. Samotný obsah je nicméně opět generován náhodně. Obdobně ale jako u solí l_p a l_z je délka soli hodnota celého čísla na intervalu $\langle 111; 366 \rangle$ a je složena z fixní hodnoty 111 a hodnoty l_{vn}. Hodnota l_{vn} je celé číslo na intervalu $\langle 0; 255 \rangle$, jelikož ji ovlivňuje hodnota 1 bajt *IV*, který určen na základě vstupních parametrů. Výpočet výsledné délky soli pro „nasolení“ je určena následující rovnicí.

$$l_n = 111 + l_{vn}$$

Hodnota l_{vn} není ovlivněna náhodnými daty, ale tajnými daty od uživatele – klíčová data a nastavení vstupních parametrů. Ačkoliv hodnota l_n není náhodná, tak hodnoty l_p a l_z jsou odvozeny na základně náhodnosti, tudíž lze tvrdit, že celková délka $l_š$ se chová náhodně a je proměnlivá při každém spuštění.

Hodnota délky $l_š$ je tedy celé číslo na intervalu $\langle l_o + 333;\ l_{op} + 1098 \rangle$. Za předpokladu šifrování prázdných otevřených dat uživatelem bude šifrový text minimálně 333 bajtů až 1098 bajtů dlouhý, kdy se tato hodnota mění náhodně.

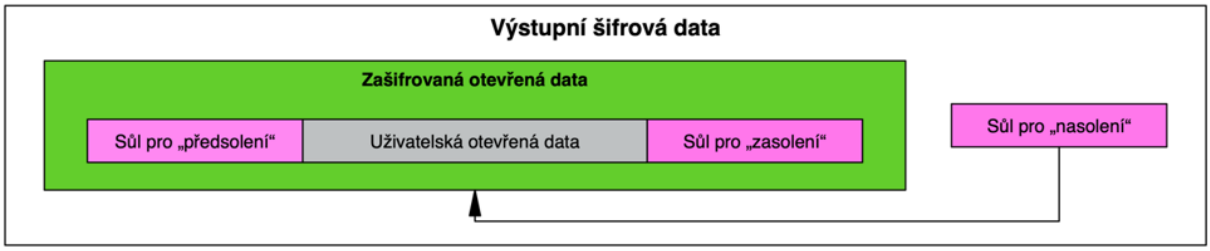
Z toho plyne důsledek, že i kdybychom měli šifrový text s délkou 500 bajtů, tak nemůžeme říci, jestli vstupní otevřená data od uživatele byla prázdná či nikoliv. O délce otevřených uživatelský dat můžeme na základě délky šifrového textu tvrdit následující:

- Byla s jistotou šifrována prázdná uživatelská otevřená data, a to pouze pokud byla výsledná hodnota $l_š = 333$.
- Nebyla s jistotou šifrována prázdná uživatelská otevřená data, a to za předpokladu, že hodnota $l_š$ byla alespoň více než 1098 bajtů.
- Při délce $l_š$ na intervalu $\langle 334; 1098 \rangle$ mohla či nemusela být délka l_{op} nulová.
- Celkově lze zpětně hodnotu l_{op} vyjádřit následující rovnicí

$$l_{op} = l_š - (l_p + l_n + l_z)$$

Tudíž hodnota l_{op} je hodnota $l_š$, která je ponížena minimálně hodnotou 333 a maximálně hodnotou 1098. Přesná hodnota ponížení je náhodná a závislá od tajných dat, tudíž nelze s přesností určit původní délku l_{op} z výsledné délky $l_š$.

Výsledná šifrová data a jejich obsah lze znázornit následujícím diagramem.



Obrázek 7 - Výsledná šifrová data (vlastní)

Z diagramu lze vidět, že výsledný obsah šifrových dat je ještě ovlivněn solí, která byla použita k „nasolení“ *IV*. Typicky pokud je při šifrování blokovými šiframi využita sůl, tak se připojuje fixně zepředu, před výstupní šifrová data. V rámci vytvořeného systému TSS je pozice soli použité pro „nasolení“ variabilní a odvozená od tajných vstupních parametrů uživatelem, včetně tajných klíčových dat. Jak je vypočtena pozice soli je podrobně rozvedeno v kapitole 7.2.5.

Nicméně lze teoreticky tvrdit, že výsledná šifrová data mají vysokou míru entropie. Hledání náhodně generované soli pro „nasolení“ je obdobné jako hledání „náhodných“ dat v „náhodných“. Výsledná entropie šifrových dat je rozebrána v kapitole 8 → „Testováním kryptografického systému TSS“.

6.3.4 Délka aktuálně šifrovaného bloku

Délka aktuálně šifrovaného bloku je v rámci navrženého systému TSS variabilní a proměnlivá na základě vstupních dat a parametrů. V porovnání s ostatními systémy, které jsou aktuálně využívané. Systém je schopen šifrovat bloky o délce 1 bajt až po délky 111 bajtů. Teoreticky by s úpravou délky klíče byl systém schopen šifrovat bloky o libovolné délce. Nicméně se zvolenou délkou

klíče 111 bajtů souvisí i maximální délka bloku, který je možné systémem najednou šifrovat.

Délka aktuálně šifrovaného bloku se variabilně mění, a to v souvislosti nejen se vstupním nastavením a podmínkami, ale také je vypočtena vždy separátně pro každý blok dat. To znamená, že před šifrováním následujícího bloku dat dojde k přepočtu hodnoty délky bloku, viz kapitola věnovaná parametrizaci 7.4.2

Délku šifrovaného prvního bloku přitom určují jiné vlastnost než délku následujících bloků. Kdy je poslední blok brán jako zbytek, které je menší jako nejmenší možná vypočtena délka bloku pro šifrování. Označme si hodnoty následovně, kde všechny hodnoty jsou celá čísla.

- Minimální hodnotu délky šifrovaného prvního bloku – lpr_{min}
- Maximální hodnotu délky šifrovaného prvního bloku – lpr_{max}
- Minimální hodnotu délky aktuálně šifrovaného bloku. – la_{min}
- Maximální hodnotu délky aktuálně šifrovaného bloku – la_{max}
- Hodnotu délky prvního šifrovaného bloku – lpr
- Hodnotu délky aktuálně šifrovaného bloku – la

Podrobně je výpočet délky aktuálně šifrovaného bloku rozebrán v kapitole 7.4.2. Pro hodnoty lpr_{min}, lpr_{max}, la_{min}, la_{max} platí následující.

$$lpr_{min} = la_{min}$$

$$lpr_{max} = la_{max}$$

Hodnoty la a lpr jsou na sobě nezávislé a jsou přepočítané před každým šifrovaným blokem dat. Každopádně lze říci, že se jedná o celá čísla na intervalech $\langle lpr_{min};\ lpr_{max}\rangle$ a $\langle la_{min};\ la_{max}\rangle$. Konkrétní hodnoty lpr a la jsou závislé na hodnotě parametru udávajícího variabilitu délky bloku, viz kapitola 7.4.2. Variabilita délky bloku může nabývat hodnotu celého čísla z intervalu $\langle 0;\ 20\rangle$. Tudíž hodnoty lpr_{min} a la_{min} nabývají hodnotu 91 a hodnoty lpr_{max} a la_{max} nabývají hodnotu 111.

Jak lze vidět z výše uvedených diagramů, tak je hodnota lpr závislá na „nasoleném“ IV a lze tedy tvrdit, že i hodnota lpr bude nabývat hodnot z intervalu $\langle 91; 111\rangle$ náhodně a bude se měnit nezávisle na vstupních datech od uživatele.

Obdobné lze tvrdit o délce aktuálně šifrovaného bloku la, která je nejenom odvozena od hodnoty udaných hodnot na začátku, ale i na základě všech předchozích stavů, z kterých je vypočten aktuální klíč. Tudíž nejen na základě hodnoty náhodně vygenerované soli k „nasolení“, ale i na základě obsahu

náhodně vygenerované soli k „předsolení“, viz závislosti aktuálního klíče v následující podkapitole. Proto lze tvrdit, že i každá následující délka *la* je náhodná a průběžné délky la se mění nezávisle na uživatelských vstupních datech.

Důsledkem je a lze tvrdit, že hodnotu *lpr* a průběžné hodnoty la nelze určit bez znalosti tajných vstupních uživatelských dat – nastavení parametrů a klíčových dat. Tyto hodnoty se chovají náhodně při každém spuštění šifrovacího procesu i v případě, že klíčová data a nastavení parametrů uživatelem zůstanou nezměněny.

6.3.4.1 Počet šifrovaných bloků

V souvislosti s výše uvedeným lze konstatovat, že pokud máme délku otevřených dat → l_o, tak počet šifrovaných bloků je náhodný, kdy minimální počet šifrovaných bloků (označme si jako psb_{min}) a maximální počet šifrovaných bloků (označme si jako psb_{max}) lze vyjádřit ve vztahu k l_o následujícími rovnicemi.

$$psb_{min} = \left\lceil \frac{l_o}{111} \right\rceil$$

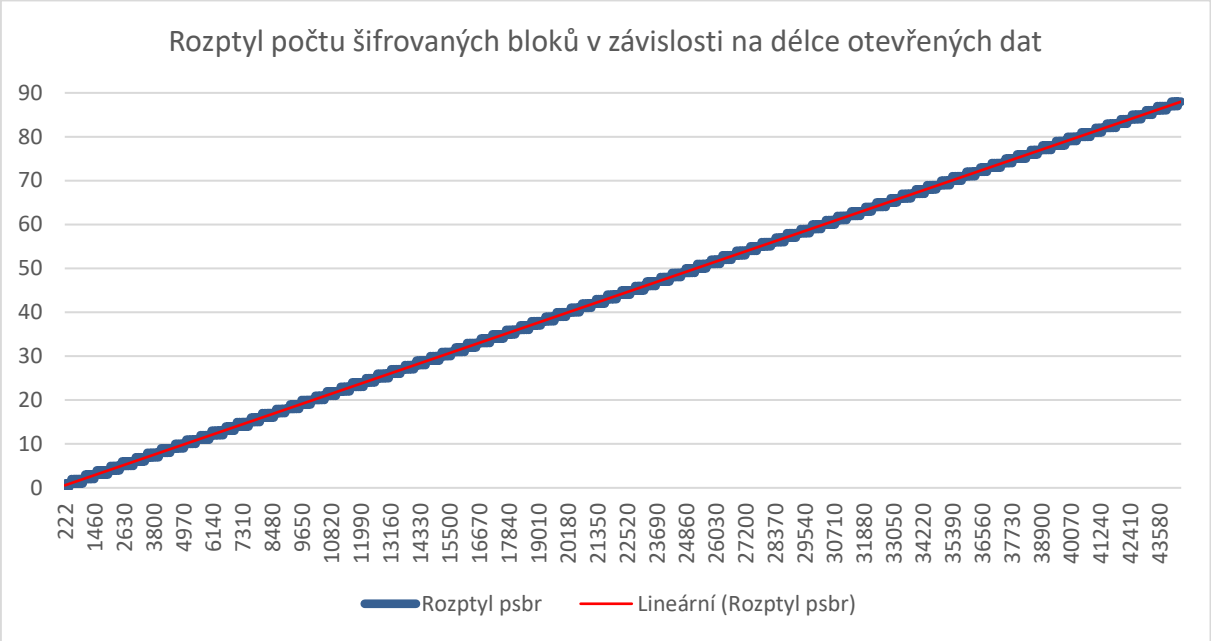
$$psb_{max} = \left\lceil \frac{l_o}{91} \right\rceil$$

Využíváme horní celou část, protože i zbytek v rámci TSS lze počítat jako za šifrovaný blok, i když je kratší než 91 bajtů. Počet šifrovaných bloků v rámci délky l_o se bude náhodně pohybovat na intervalu $\langle psb_{min};\ psb_{max} \rangle$. Dále můžeme říci, že čím bude délka otevřených dat vyšší, tím bude i počet šifrovaných bloků moci nabývat více hodnot. Neboli rozptyl počtu šifrovaných bloků poroste. Pro ukázku si zvolíme délky l_o rovny 11111 bajtů a 111111 bajtů. V případě délky l_o = 11111 bajtů se po dosazení do výše uvedených rovnic bude počet šifrovaných bloků pohybovat na intervalu $\langle psb_{min};\ psb_{max} \rangle$, tudíž na intervalu $\langle 101;\ 123 \rangle$. Pokud bychom šifrovali otevřená data o délce 111111 bajtů, tak se počet šifrovaných bloků bude pohybovat na intervalu $\langle 1001;\ 1221 \rangle$. Pokud bychom si označili počet aktuálně šifrovaných bloků označili hodnotou psb_a, lze tvrdit, že hodnota psb_a se bude pohybovat na intervalu $\langle psb_{min};\ psb_{max} \rangle$. V souvislosti s tímto intervalem si můžeme označit hodnotu psbr jako rozptyl hodnot počtu šifrovaných bloků. Rozptyl počtu šifrovaných bloků lze vyjádřit následující rovnicí.

$$psb_r = psb_{max} - psb_{min}$$

V souvislosti s délkou l_o 11111 bajtů můžeme určit psb_r roven 22 a pro hodnotu l_o rovné 111111 bajtů bude rozptyl psb_r roven 220. Při zobecnění a

z reprezentace následujícího grafu lze říci, že rozptyl psb_r je lineárně závislý na délce l_o.



Obrázek 8 - Rozptyl počtu aktuálně šifrovaných bloků vyjádřený vůči délce otevřených dat (vlastní)

Z výše uvedeného grafu lze vidět, že závislost rozptylu psb_r na délce šifrových dat l_o je lineární, kdy červená přímka odpovídá lineární spojnici trendu.

Jelikož hodnota l_o souvisí i s délkami solí pro „předsolení“ a „zasolení“, které souvisí s náhodnou solí pro „nasolení“, tak i kdybychom šifrovali stejná otevřená data, stejným klíčem pomocí stejně nastavených uživatelských parametrů, tak i počet šifrovaných bloků se bude náhodně měnit. Fakt, že se počet šifrovaných bloků mění náhodně nezávisle na vstupních uživatelských datech, poskytuje ochranu před útoky založenými na časování, protože doba šifrování úzce souvisí s počtem šifrovaných bloků.

6.3.5 Aktuální klíč

Šifrovací klíč je vypočten vždy samostatně pro účely šifrování aktuálního bloku otevřených dat. Aktuální klíč je polymorfně odvozen nejen na základě tajných vstupních uživatelských dat → nastavení parametrů a uživatelská klíčová data, ale v případě prvního bloku i na základě soli pro „nasolení“ a pak následně je aktuální klíč odvozen dle posledních šifrových dat, předchozích otevřených dat (včetně soli pro „předsolení), předchozího klíče a předchozích parametrů šifrování. Režimy činnosti pro odvození aktuálního klíče jsou podrobně rozebrané v kapitole 7.5.

Jelikož je první klíč „nasolen“ pomocí soli (náhodná data) a další klíče jsou závislé na předchozích datech, kterých je součástí sůl pro „předsolení“ (náhodná data), tak lze aktuální klíč považovat za náhodný. To znamená, že při každém šifrování se klíče mění v souvislosti s náhodně generovanými solemi → „předsolení“ a „nasolení“. A to i v případě, že nedošlo ke změně otevřených uživatelských dat nebo klíčových dat nebo nastavení parametrů uživatelem.

Aktuální klíč má vždy délku 111 bajtů, a to i v případě, že je šifrován blok otevřených dat kratší než 11 bajtů. V tomto případě se k šifrování používá jen odpovídající počet bajtů klíče v závislosti na délce šifrovaného bloku. Aktuálně se využívá vždy první bajty klíče, to znamená, že pokud je šifrován blok otevřených dat o délce 98 bajtů, tak je použito prvních 98 bajtů klíče. V rámci budoucího vývoje je v plánu tyto bajty vybírat variabilně dle parametrizace uživatelem. Při šifrování bloku 98 bajtů je při odvozování klíče pro následující blok modifikované opět 98 bajtů klíče. Což vede k tomu, že před šifrováním dalšího bloku dat je vždy s jistotou upraveno prvních 91 bajtů klíče (nejmenší délka šifrovaného bloku) a zbývajících 20 bajtů je změněno pouze pokud jsou šifrované bloky delší. V souvislosti se začleněním náhodných dat do procesu lze říci, že 20 posledních bajtů jsou měněny náhodně a zda budou či nebudou bajty změněny je proměnlivé nezávisle na uživatelském vstupu.

Pro shrnutí můžeme říct, že první klíč (slouží po zašifrování soli pro „předsolení“) závisí na:

- Tajných uživatelských klíčových datech → vygenerované IV
- Uživatelských tajných vstupních parametrech
- Náhodná sůl pro „nasolení“
- Defaultní substituční tabulce
- V případě režimu činnosti PM-DC-LM na aktuálním stavu generátoru

Klíč pro šifrování prvního bloku uživatelských otevřených dat a další všechny klíče jsou závislé na všem předchozím, což konkrétně znamená:

- Předchozí šifrovací klíč
- Předchozí otevřená data
- Včetně náhodné soli pro „předsolení“
- Předchozí šifrová data
- Přepočítané parametry
- V případě režimu činnosti PM-DC-LM na aktuálním stavu generátoru

6.3.6 Počet rund

Počet rund v rámci systému TSS je v porovnání s aktuálně využívanými blokovými šiframi nekonstantní a relativně malý. Počet rund je odvozen pro každý šifrovaný blok zvlášť na základě parametrizace a aktuálního klíče. Počet rund, kterým může být aktuální blok otevřených dat šifrovaný je celé číslo na intervalu $\langle 1; 4 \rangle$. Runda v rámci TSS znamená využití všech tří šifrovacích operací, kdy pořadí operací se mění v souvislosti s aktuálním šifrovacím klíčem. Odvození pořadí operací je v porovnání s diplomovou práci autora také upraveno. Pro porovnání viz následující dva obrázkové reprezentace zdrojových kódů.

```
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 9 - Vygenerování pořadí šifrovacích operací pro rundy; diplomová práce autora (vlastní)

```
def VygenerujSledOperaci(klic, pocetRund, x):

    sledOperaci = ''
    for i in range(0, pocetRund):
        zbytek = klic[(x + i) % len(klic)] % 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 10 - Vygenerování pořadí operací šifrování v rámci systému TSS (vlastní)

Z výše uvedených zdrojových kódů lze vidět hlavně modifikaci v podobě parametrizaci → závislost na proměnné X (rozepsaná v kapitole 7.4.5). Lze vidět, že pořadí operací souvisí s parametrizací a na počtu bajtů klíče v závislosti na počtu rund. Bajty klíče i pozice bajtů v rámci klíče pro odvození pořadí operací jsou polymorfně proměnlivé.

Počet rund, jakožto i pořadí operací v rámci rund při šifrování jsou variabilně voleny. Souvisí opět nejen na aktuálním klíči a parametrizaci, ale v rámci celého procesu i na vygenerovaných náhodných dat ve formě solí pro „nasolení“ i „předsolení“. Před šifrováním není možné říct, kolik rund bude využito k šifrování a nelze ani určit pořadí operací v rámci šifrování prvního či dalšího bloku otevřených dat. To opět ani když nebudou změněna uživatelská data → klíčová data, otevřená data či uživatelské vstupní parametry.

6.3.7 Průběh aktuálního šifrování

Průběh aktuálního šifrování souvisí s popsaným v předchozí kapitole → pořadím operací, počtem rund, ale souvisí s tím, že samotné operace pro šifrování jsou závislé na klíči (viz diplomová práce autora a popis v rámci kapitoly 7.1).

Jelikož lze považovat při zahrnutí náhodných dat ve formě vygenerovaných solí počet rund, pořadí operací i klíč za náhodné, tak celkový proces šifrování v rámci systému TSS lze považovat za náhodný. Opět i když uživatel použije stejná klíčová data a stejné uživatelské vstupní parametry pro šifrování stejných otevřených dat.

6.3.8 Shrnutí vlastností polymorfního kryptografického systému TSS

Na základě předchozích kapitol lze tvrdit, že všechny vlastnosti, součásti i chování kryptografického systému TSS jsou polymorfní a chovají se polymorfně, a to nejen v souvislosti na uživatelském vstupu, ale i ve vazbě na náhodně vygenerovaná data → soli. Celý proces šifrování a kryptografický systém TSS je tudíž náhodně polymorfní, kdy pro každý šifrovaný blok je proces modifikován. Pro shrnutí jsou následující vlastnosti a součásti TSS polymorfní a náhodné:

- Odvození IV → dle klíčových dat
- Délka šifrovaného bloku
- Klíče pro šifrování
    - Režimy činnosti, včetně generátoru (PM-DC-LM)
- Počet rund
- Pořadí operací
- Operace šifrování
- Solení → podrobně viz kapitola 7.2
    - Délky solí pro „předsolení“, „zasolení“ a „nasolení“
    - Proces solení
    - Pozice soli pro „nasolení“ v rámci šifrových dat
- Substituční tabulka

7 NÁVRH POLYMORFNÍCH SKTRUKTUR BLOKOVÝCH ŠIFER

Kapitola se věnuje jednomu ze stěžejních cílů disertační práce a v rámci doktorského studia mu byla věnována největší pozornost. V rámci předchozí kapitoly by předveden komplexní systém jako celek. V této kapitole bude rozebrán výzkum a návrh jednotlivých části systému v souladu s terminologií problematiky návrhu blokových šifer. Zároveň byly všechny části navrženy v souladu s tématem práce tak, aby bylo vše co nejvíce polymorfní – závislé na aktuálních podmínkách a vstupních datech.

V rámci problematiky návrhu komplexního polymorfního šifrovacího systému byly řešeny následující části blokových šifer a šifrování dat; tyto části se běžně využívají při šifrování pomocí blokových šifer či při návrhu blokových šifer:

- Odvození a tvorba inicializačního vektoru z tajných dat uživatele
- Bloková šifra – samotné šifrovací funkce
- Správa klíčů
- Režim činnosti blokových šifer
    - Proces zahrnutí entropie – solení

Dále v rámci doktorského studia proběhl návrh parametrizace systému a vlastností šifrování, které běžně v kryptografii není řešeno, viz kapitola věnující se současnému stavu řešené problematiky.

7.1 Návrh a vylepšení blokové šifry

Prvotní návrh blokové šifry → šifrovacích operací pro šifrování jednotlivých bloků otevřených dat je podrobně rozebráno v rámci diplomové práce autora [11]. V rámci diplomové práce je podrobně rozepsané fungování blokové šifry, tak i testování. V disertační práci budou v rámci této kapitoly rozepsány změny. Celkově nebylo nutné příliš operace a blokovou šifru měnit, protože už v rámci diplomové právě byla bloková šifra i její operace navrženy v souladu s principem polymorfního chování.

V rámci blokové šifry můžeme počítat následující operace:

- Operace substituce
- Operace přičtení klíče
- Operace mixování (transpozice)
- Funkce pro zamíchání substituční tabulky klíčem
- Funkce pro výpočet pořadí operací v rámci rund

7.1.1 Operace substituce

Jak lze dohledat v rámci diplomové práce autora, tak substituce v rámci šifry TSS probíhá v závislosti na klíči, který je určen pro šifrování otevřených dat aktuálního bloku. Tudíž se nejedná o klasickou substituci jednoho bajtu za druhý, kdy by výsledná hodnota byla určena vstupním bajtem, ale v závislosti na klíči může být vstupní bajt substituován libovolnou hodnotou na intervalu $\langle 0; 255 \rangle$. Přihlédneme-li k faktu, že klíč je odvozen od vstupních parametrů, tajných klíčových dat, a hlavně na tvaru náhodných solí, tak lze říci, že i substituce se chová náhodně. To znamená, že se průběh substituce mění v souvislosti se spuštěním šifrování. I za předpokladu, že nedojde ke změně vstupních otevřených dat nebo klíčových dat nebo nastavení parametrů uživatelem. Toto lze brát jako hlavní vylepšení v rámci návrhu kryptografického systému TSS → vytvořený solící systém. V rámci disertační práce došlo k následujícím změnám:

- Vliv náhodných solí na proces substituce
- Mírná modifikace implementace pro využití variabilní délky bloků
- Drobná úprava samotné operace → vliv pozice aktuálně šifrovaného bajtu otevřených dat
- Optimalizace kódu

Následující obrázky ukazují původní implementaci z diplomové práce v porovnáním s aktuální implementací.

```
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 11 - Původní implementace substituce - diplomová práce (vlastní)

```
def Substituce(blokKSifrovani, klic, substitucniTabulkaIn, delkaBloku):

    zasifrovanyBlok = bytearray(b'')
    for i in range(0, delkaBloku):
        index = (blokKSifrovani[i] + klic[i] + i) % 256
        zasifrovanyBlok.append(substitucniTabulkaIn[index])
    return bytes(zasifrovanyBlok)
```

Obrázek 12 - Aktuální implementace substituce - systém TSS (vlastní)

Z obrázku lze vidět, že substituce byla změněna o začlenění vlivu pozice šifrovaného bajtu v rámci výpočtu indexu pro nalezení znaku v rámci substituční tabulky. V podstatě o konstantní posun pozice znaku v substituční tabulce v závislosti na pozici aktuálně substituovaného bajtu otevřených dat. Stejným způsobem byla upravená operace pro zpětnou substituci při dešifrování bloku otevřených dat.

7.1.2 Operace přičtení klíče

Obdobně u operace přičtení klíče jako u substituce lze říci, že v rámci diplomové práce byla navržena polymorfně, protože přičtení klíče se odvíjí od samotných hodnot bajtů klíče. Zároveň díky začlenění náhodně generovaných solí do procesu dochází k přičtení klíče náhodných způsobem. Lze říci, že došlo k následujícím změnám:

- Vliv náhodných solí na proces přičtení klíče
- Mírná modifikace implementace pro využití variabilní délky bloků
- Drobná úprava samotné operace → vliv pozice aktuálně šifrovaného bajtu otevřených dat
- Optimalizace kódu

Následující obrázky zachycují změny provedené v aktuální implementaci operace přičtení klíče v porovnání s tvarem v diplomové práci.

```
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 13 - Původní implementace přičtení klíče - diplomová práce (vlastní)

```
def Add(blokKSifrovani, klic, delkaBloku):

    zasifrovanyBlok = bytearray(blokKSifrovani)
    for i in range(0, delkaBloku):
        aktZnakKlice = klic[i]
        if aktZnakKlice % 2 == 0:
            for j in range(i, delkaBloku):
                zasifrovanyBlok[j] = (zasifrovanyBlok[j] + aktZnakKlice + i) % 256
        else:
            for j in range(i, -1, -1):
                zasifrovanyBlok[j] = (zasifrovanyBlok[j] + aktZnakKlice + i) % 256
    return bytes(zasifrovanyBlok)
```

Obrázek 14 - Aktuální implementace přičtení klíče - systém TSS (vlastní)

Patřičně v závislosti na pozici aktuálně přičítaného bajtu klíče byla upravena operace pro zpětné odečtení klíče ze šifrových dat.

7.1.3 Operace mixování (transpozice)

Z původních tří operací pro šifrování nebyla operace pro mixování (transpozici) nikterak principiálně pozměněna v porovnání s implementaci v diplomové práci autora. Jediná změna souvisí s optimalizací implementace a zdrojového kódu. Úpravy a optimalizace jsou patrné z následujících obrázků.

```
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 15 - Původní implementace mixování (transpozice) - diplomová práce (vlastní)

```
def Mixovani(blokKSifrovani, klic, delkaBloku):

    zasifrovanyBlok = bytearray(blokKSifrovani)
    for i in range(0, delkaBloku):
        mixIndex = klic[i] % delkaBloku
        zasifrovanyBlok[i], zasifrovanyBlok[mixIndex] = zasifrovanyBlok[mixIndex], zasifrovanyBlok[i]
    return bytes(zasifrovanyBlok)
```

Obrázek 16 - Aktuální implementace mixování (transpozice) - systém TSS (vlastní)

Za jedinou změnu, která nesouvisí s tvarem operace mixování, lze považovat závislost operace na délce bloku, která je v porovnání s návrhem v diplomové práci v rámci systému TSS polymorfní a náhodná. Proto v rámci diplomové práce byla délka operace pro mixování konstantní, ale v rámci TSS

bude časově závislá na délce bloku. Časová závislost se bude měnit i pro operace substituce a přičtení klíče.

7.1.4 Funkce pro zamíchání substituční tabulky

Jak je patrné z diplomové práce autora, tak substituční tabulka je proměnlivá a není statická. A v rámci šifrování každého bloku otevřených dat je využitá jiná substituční tabulka, která je zamíchaná na základě předchozího šifrovacího klíče.

```
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 17 - Původní implementace funkce pro zamíchání substituční tabulky - diplomová práce (vlastní)

```
def ZamichaniSubstitucniTabulky(substitucniTabulkaIn, klic):

    substitucniTabulkaOut = substitucniTabulkaIn[:]
    l = len(klic)
    for i in range(0, 256):
        substitucniTabulkaOut[i], substitucniTabulkaOut[klic[i % l]] = substitucniTabulkaOut[klic[i % l]], substitucniTabulkaOut[i]
    return substitucniTabulkaOut
```

Obrázek 18 - Aktuální implementace funkce pro zamíchání substituční tabulky - systém TSS (vlastní)

Jak je možné vidět z obrázků výše, tak podobně jako u operace mixování došlo v porovnání s implementací diplomové práce pouze k optimalizaci zdrojového kódu.

7.1.5 Funkce pro výpočet pořadí operací

Výpočet pořadí operací a popis změn v porovnání s diplomovou prací autora je rozveden v rámci kapitoly 6.3.6.

7.2 Princip solení a zahrnutí entropie do procesu šifrování

V průběhu tvorby kryptografické systému a při jeho výzkumu se projevovaly nedostatky v difúzi při šifrování. V závislosti na změnách v rámci otevřených dat bylo možné pozorovat konkrétní změny v šifrových datech. Toto nastávalo při změně pouze otevřených dat. Za další nedostatek bylo možné považovat přílišnou fixaci průběhu šifrování na vstupní podmínky → otevřená data, klíčová data a vstupní nastavení parametrů. Již ve variantě kryptografického systému bez využití solí bylo možné pozorovat polymorfní chování v souvislosti

se vstupním nastavením, což potvrzují výsledky testování v rámci kapitoly 8.1. Nicméně, zvýšení difúze šifrovacího systému byl jeden ze základních požadavků, který bylo nutné vyřešit.

Nejprve byl navržen systém solení, který využívá náhodných dat ve formě soli připojených na začátek otevřených dat. Tato sůl měla pevnou délku 111 bajtů. Tudíž v první fázi kryptografický systém obsahoval pouze jednu sůl. Což vedlo k markantnímu nárůstu difúze systému a odstranění závislosti změn otevřených dat na změnách výstupních šifrových dat. V konečném důsledku i k takové difúzi, že beze změny vstupních dat a parametrů dochází ke změně na šifrových datech. Kromě zvýšení difúze systému také došlo k vysoké polymorfizaci systému. Jelikož je celý systém postaven tak, aby docházelo ke změnám šifrování a parametrů ve vazbě ke změnám otevřených dat, tak došlo ke změně systému v náhodně polymorfní. Využití soli na začátku bylo označeno jako „předsolení“ a je podrobněji rozepsané v podkapitole 7.2.2.

V druhé fázi byl systém rozšířen o druhou sůl, která ve formě náhodně generovaných dat slouží k „nasolení“ klíče. „Nasolení“ klíče znamená postupné přičtení a modulování bajtů klíče s bajty soli. Proces „nasolení“ je rozebrán v podkapitole 7.2.4. Opět byla nejprve sůl fixní délky 111 bajtů. Cílem druhé soli bylo zvýšení konfúze systému a zároveň další zvýšení náhodnosti polymorfního chování systému → bez nutnosti změny tajných klíčových dat dochází ke změně systému a šifrovacích klíčů pro jednotlivé bloky otevřených dat. Jelikož proces šifrování otevřených dat a tvorba klíčů pro další bloky jsou založeny nejen na předchozích otevřených datech, ale i na předchozím šifrovacím klíči a předchozích šifrových datech. V rámci vývoje „nasolení“ bylo potřeba vyřešit problém s přenosem soli. V praxi je typicky sůl přenášena jako samostatný vstupní parametr nebo je připojena na začátek výstupních šifrových dat. Systém TSS tuto problematiku řeší algoritmickým „včleněním“ soli pro „nasolení“ do šifrových dat. Způsob a popis je podrobněji rozebrán v podkapitole 7.2.5.

V třetí fáze vývoje systému solení byla přidána ještě třetí sůl ve formě náhodně generovaných dat, která byla zahrnuta do procesu označeného jako „zasolení“, viz kapitola 7.2.3. Tato sůl aktuálně nemá stejný přínos jako soli pro „předsolení“ a „nasolení“, ale v souvislosti s nahrazením doplňování dat na konec otevřených dat (paddingu, viz diplomová práce autora) byla použita třetí sůl. V poslední fází, kdy byl proces solení zparametrizován, došlo díky tomu k dalšímu prohloubení nejistoty v délce otevřených dat ze znalosti šifrových dat. V rámci budoucího vývoje je v plánu využít poslední sůl jako kontrolní autentizační tag a rozšíření tak režimů činnosti o možnost autentizace a kontrolu integrity šifrových dat. Aktuální TSS nezahrnuje tuto možnost.

Poslední fáze geneze systému solení spočívala v parametrizaci systému. To znamená, že délka solí aktuálního systému TSS není fixních 111 bajtů, jako tomu bylo z počátku vývoje, ale že je závislá na nastavení vstupních parametrů, viz kapitola 7.4.4 a kapitoly 6.3.2 a 6.3.3 věnující se délce otevřených dat a šifrových dat. Dále byla vytvořena závislost délky solí pro „předsolení“ a „zasolení“ na „nasoleném“ $IV \rightarrow$ což vedlo k jejich náhodným délkám. Dalším přínosem a důsledkem parametrizace solení byla náhodnost pozice soli pro „nasolení“ v šifrových dat.

Pro budoucí vývoj je připravena myšlenka rozšíření solení o čtvrtou sůl (vznik systému Quadruply Salted Smallie – čtyřnásobně nasolené Smallie $\rightarrow$ QSS), která bude sloužit pouze pro potřeby náhodnosti pozice soli v rámci šifrových dat. Dalším vylepšení systému TSS je v plánu využití celého klíče pro výpočet pozice soli, kdy by byla sůl pro „nasolení“ po částech „vsolena“ do šifrových dat.

7.2.1 Struktura solení v rámci TSS

Jak bylo popsané v úvodu kapitoly, tak systém solení TSS využívá tří solí, které mohou nabývat délek 111 až 366 bajtů. Proces solení je závislý na parametrizaci $\rightarrow$ tajných uživatelských vstupních parametrech, na tajných klíčových datech, ale je ovlivněn i samotnou náhodností generovaných solí navzájem. Systém solení spočívá v následujících částech/funkcích:

- Parametrizace solení
- Generování a využití soli pro „předsolení“
- Generování a využití soli pro „zasolení“
- Generování a využití soli pro „nasolení“
- Výpočet pozice soli pro „nasolení“ v šifrových datech

7.2.2 Proces „předsolení“

Proces „předsolení“ a jeho vliv na systém je z velké části popsán v kapitolách 6.3.2, 7.4.4 a v úvodu této kapitoly. Pro shrnutí, generování soli – její délka, je ovlivněna následujícími faktory:

- Nasoleným IV
- Vstupními uživatelskými parametry
  - Defaultní délka „předsolení“
  - Defaultní řídící proměnná pro výpočet délky „předsolení“

Jak bylo popsané v kapitole 6.3.2, tak sůl pro „předsolení“ může nabývat délek na intervalu $\langle 111; 366 \rangle$. V základu je tedy jisté, že délka soli bude minimálně 111 bajtů a o kolik delší bude je vypočteno na základě „nasoleného“

IV. Pomocí řídící proměnné pro výpočet délky soli pro „předsolení“ se zvolí pozice bajtu „nasoleného klíče“ a na základě defaultní délky soli pro „předsolení“ se vypočte výsledná délka soli. Výpočet lze zachytit rovnicí popsané v kapitole 7.4.4.

7.2.3 Proces „zasolení“

Proces „zasolení“ a jeho vliv na systém je také z velké části popsán v kapitolách 6.3.2, 7.4.4 a v úvodu této kapitoly. Pro shrnutí, generování soli – její délka, je ovlivněna následujícími faktory:

- Nasoleným IV
- Vstupními uživatelskými parametry
    - Defaultní délka „zasolení“
    - Defaultní řídící proměnná pro výpočet délky „zasolení“

Jak bylo popsané v kapitole 6.3.2, tak sůl pro „zasolení“ může nabývat délek na intervalu $\langle 111; 366 \rangle$. Opět můžeme tvrdit, že délka soli bude minimálně 111 bajtů a zbývající část je vypočtena z „nasoleného“ *IV*. Pomocí řídící proměnné pro výpočet délky soli pro „zasolení“ je vybrán bajt „nasoleného klíče“ a spolu s defaultní délkou soli pro „předsolení“ se vypočítá výsledná délka soli. Konkrétní rovnice pro výpočet délky soli systému TSS lze nalézt v kapitole 7.4.4.

7.2.4 Proces „nasolení“

Proces generování soli a „nasolení“ je v porovnání s využitím solí pro „předsolení“ a „zasolení“ nejsložitější. V této podkapitole si popíšeme generování soli s určením délky a dále i postup „nasolení“ *IV* pomocí vygenerované soli. Vypočet pozice soli a princip vmísení soli do výsledných šifrových dat je rozebraný samostatně v následující podkapitole.

Sůl pro „nasolení“ může nabývat stejných hodnot délky jako v případě solí pro „předsolení“ a „zasolení“ $\rightarrow \langle 111; 366 \rangle$. Délka je ovlivněna také na základě vstupních uživatelských parametrů, ale na rozdíl od dvou zmíněných solí její délka závisí na původním *IV*. Shrneme-li to, tak délka soli pro „nasolení“ je vypočtena na základě těchto dat:

- IV
- Vstupními uživatelskými parametry
    - Defaultní délka „nasolení“
    - Defaultní řídící proměnná pro výpočet délky „nasolení“

Pro výpočet délky je použita stejná rovnice jako pro výpočet délky „předsolení“, viz kapitola 7.4.4. Kromě zvýšení zakrytí délky výsledných

šifrových dat má délka soli druhý význam. Jelikož je šifrovací klíč (*IV*) dlouhý 111 bajtů, tak i sůl pro „nasolení“ je minimálně 111 bajtů dlouhá, a tudíž dochází k „nasolení“ každého bajtu *IV* alespoň jednou. Alespoň jednou, protože délka soli může být až 366 bajtů v závislosti na parametrizaci a *IV*. To znamená, že prvních 33 bajtů může být „nasoleno“ i v krajním případě čtyřnásobně. Kolikrát byly bajty IV „nasoleny“ nelze ze znalosti šifrových dat určit. V budoucnu by bylo možné využít další parametry a čtvrtou sůl pro generování soli pro „nasolení“ a například pozice bajtu v *IV*, odkud by se *IV* „nasolilo“.

Nasolení probíhá prostým přičtení bajtu *IV* s bajtem soli a modulací 256. Proces „nasolení“ lze vidět z následujícího zdrojového kódu, který je součástí implementace systému TSS.

```
def NasoleniIV(IV, sul):

    bIV = bytearray(IV)
    bSul = bytearray(sul)
    for i in range(0, len(bSul)):
        bIV[i % 111] = (bIV[i % 111] + bSul[i]) % 256
    return(bytes(bIV))
```

Obrázek 19 - Proces "nasolení" IV pomocí vygenerované soli

7.2.5 Výpočet pozice soli v šifrových datech

Jak už bylo zmíněno výše, tak se sůl ve formě náhodně generovaných dat využívá v praxi, a to zejména pro úpravu dat pro odvození šifrovacího klíče. Je to vhodné zejména z důvodu, pokud je šifrovací klíč odvozován například z hesla apod. Zde sůl slouží pro zvýšení míry entropie a náhodnosti, a ve výsledku i kvality šifrovacího klíče. Každopádně, pokud je v praxi sůl využívaná, tak je nutné sůl přenášet spolu se šifrovými daty → obvykle jako první část šifrových dat. To umožní výpočet šifrovacího klíče z tajných dat a následně dešifrování dat. Například využití v rámci systému TrueCrypt [35] nebo aktuálně VeraCrypt [36][3],

[3] *Formát šifrových dat zde –*

kdy víme, že vždy prvních 64 bajtů šifrových dat je sůl. Dalším nástrojem, který při šifrování využívá sůl a připojuje ji k šifrovým datům je nástroj „Paranoia Encryption“[4]. O těchto nástrojích můžeme říci, že využívají „nasolení“ šifrovacího klíče pomocí algoritmů uvedených v kapitole 2.5.

Lze tvrdit, že tento způsob soli je fixní a pokud má útočník přístup k šifrovým datům, tak má i přístup k využité soli, pokud zná specifikaci nástrojů. Tudíž nedochází k žádnému vylepšení bezpečnosti. V rámci disertační práce bylo cílem navrhnout systém solení, který by měl následující vlastnosti:

- Využití soli pro posílení klíče → vytvořené „nasolení“.
- Eliminaci nutnosti přenášet sůl se šifrovacím klíčem → závislost na asymetrické kryptografii.
- Otevřenost systému a zároveň znemožnění lokalizace soli v šifrových datech → parametrické a polymorfní vmísení soli do šifrových dat (viz dále) závislé na *IV*.

V rámci disertační práce byly stanoveny celkem tři způsoby, jak začlenit sůl pro „nasolení“ do šifrových dat pomocí *IV*. Všechny tři způsoby jsou parametrizovány na základě uživatelsky zvolených parametrů. Pro shrnutí, všechny tři metody jsou ovlivněny následujícími faktory:

- Vypočtené *IV* na základě uživatelských tajných klíčových dat → využití jedenácti bajtů klíče dle parametrů (viz kapitola 7.4.4)
- Volba uživatelských parametrů (viz kapitola 7.4.4)
- Délka otevřených dat včetně solí pro „předsolení“ a „zasolení“

---

https://veracrypt.fr/en/VeraCrypt%20Volume%20Format%20Specification.html

[4] *https://paranoiaworks.mobi/pte/specifications.html*

Lze tvrdit, že první metoda byla následně optimalizována v druhou metodu, která byla následně optimalizována ve výslednou metodu číslo tři, kdy poslední metoda je využívaná v systému TSS.

7.2.5.1 Metoda první

Jak bylo uvedeno výše, tak odvození pozice soli bylo vypočteno na základě jedenácti bajtů *IV*. V rámci prvního způsobu byly hodnoty těchto bajtů sečteny a modulovány délkou otevřených dat. Pozice prvního bajtu *IV* byla vypočtena na základě parametrů. Velkou nevýhodou a důvodem pro optimalizaci byl fakt, že maximální hodnota, kterou můžeme ze součtu hodnot jedenácti bajtů *IV* dostat je $11 \cdot 255$, což je pouze 2805. To by vedlo k tomu, že pokud bychom měli otevřená data delší jak 2805 bajtů, tak by pozice soli v šifrových datech byla vždy do pozice 2805 bajtu (dle hodnoty součtu jedenácti bajtů *IV*). Pokud si vypočítáme maximální počet, který jsme schopni na základě hodnot jedenácti bajtů určit[5], tak je 2806 (včetně nulté) možných pozic velmi málo. V porovnání se systémy uvedenými v rámci úvodu kapitoly 7.2.5, kdy je pozice i délka soli jednoznačná, tak se jedná o významné vylepšení, a to i za předpokladu, že bychom měli sůl pro „nasolení“ pevné délky, což v rámci TSS díky parametrizaci není a může nabývat délek na intervalu $\langle 111; 366 \rangle$ bajtů. V rámci systému TSS to znamená, že by musel útočník při otevřených datech dlouhých 2805 bajtů včetně vyzkoušet $2806 \cdot 256 = 718336$ různých kombinací pozice soli a délky soli.

7.2.5.2 Metoda druhá

Vylepšení v rámci druhé metody spočívalo hlavně ve změně součtu hodnot bajtů *IV* na součin hodnot bajtů *IV*, kdy každá hodnota byla inkrementována o jedna (kvůli nulování pomocí nulových bajtů). Počet možných pozic soli v šifrových datech zde narostl a přiblížil se k maximální ideální hodnotě 256^{11}. Každopádně při bližším prozkoumání lze zjistit, že zde dochází ke kolizím, protože pozice souvisí pouze s hodnotami bajtů, nikoliv s jejich pozicí v rámci *IV*. Pro názornost, pokud bychom měli *IV*, z kterého v souvislosti s parametrizací zvolíme 11 bajtů, kdy by jejich reprezentace v hexadecimálním tvaru (viz níže), tak bude pozice soli v šifrových datech stejná.

- *00000000000000000aabb*

---

[5] *309485009821345068724781056*

- *00000000000000000bbaa*

Konkrétní počet možných pozic není jednoduchý problém pro výpočet a souvisí s problematikou prvočísel.

7.2.5.3 Metoda třetí

Nevýhodu v rámci druhé metody odstraňuje metoda třetí, kdy nedochází ani k součtu hodnot bajtů *IV*, ani k součinu hodnot, ale k jejich reprezentaci jako bitového čísla o délce 88 bitů. Těchto 88 bitů klíče je z *IV* parametricky zvoleno. Tímto způsobem docílíme maximálního počtu rozdílných hodnot v závislosti na $\text{IV} \rightarrow 256^{11} = 2^{88} = 309485009821345068724781056$ a zároveň možnosti šifrovat delší otevřená data. Výpočet indexu bajtu *IV* lze reprezentovat následujícím zdrojovým kódem.

```
def VypocetIndexuProVypocetPoziceSoli(IV, defPromenna, defRidiciPromenna):
    return (defPromenna + (IV[defRidiciPromenna])) % 100
```

Obrázek 20 - Odvození indexu prvního bajtu IV pro výpočet pozice soli (vlastní)

Výpočet samotné pozice soli pro „nasolení“ v rámci šifrových dat zobrazuje následující zdrojový kód.

```
def VypocitejPoziciSoli(IV, parametr, delka):
    return int(IV[parametr:parametr + 11].hex(), 16) % delka
```

Obrázek 21 - Výpočet pozice soli v šifrových datech (vlastní)

Jak už bylo uvedeno výše, tak třetí metoda je aktuálně nejlepší navržená a využívaná v systému TSS poslední verze.

7.2.6 Vymezení bezpečnosti solení a shrnutí

Proces solení je prakticky nezávislý na uživatelském vstupu, a proto i bezpečnost je z většiny nezávislá na uživatelských datech. Hlavní část bezpečnosti je proto spojená s kvalitou generování náhodných dat a souvisí s danou problematikou. Tudíž pokud jsou generovaná data náhodná, tak lze považovat TSS za náhodný. Ideální je využít tak zvaných „true-random“ generátorů → využití atmosférického šumu, kvantových generátorů apod. Pro praktické testování je nyní generování náhodných solí v rámci systému TSS založené na základní funkci *urandom()* v rámci jazyka Python 3.x, ale není problém využití jakéhokoliv jiného generátoru.

Jediná část v rámci solení, která nejméně souvisí s náhodnými daty a více s uživatelským vstupem je výpočet pozice soli v rámci šifrových dat. Ta, jak bylo

uvedeno v rámci podkapitole výše, souvisí zejména s klíčovými daty a vstupními uživatelskými parametry. Tato funkce je nicméně polymorfní v souvislosti s klíčovými daty a uživatelskými parametry, a tudíž je jen částečně náhodná. Za částečně náhodnou ji lze považovat, protože pozice souvisí s délkou otevřených dat, která zahrnují soli pro „předsolení“ a „zasolení“ → délky solí souvisí s nasoleným *IV*. Tedy i pozice soli v rámci šifrových dat se chová částečně náhodně. Prakticky to znamená, že i když bychom šifrovali pomocí stejného IV, pomocí stejných parametrů, tak i tak se může pozice soli měnit v souvislosti s délkami solí pro „předsolení“ a „zasolení“ → 512 různých pozic. Jak bylo naznačeno v úvody kapitoly, tak by šlo využít generování čtvrté soli, která vy sloužila pouze pro potřeby výpočty pozice soli.

Celkově tedy bezpečnost části TSS pro solení souvisí zejména s kvalitou a náhodností generovaných solí. Nicméně díky funkcionalitě solení lze považovat systém TSS za **náhodně polymorfní**.

7.3 Návrh způsobu generování šifrovacího klíče na základě tajných dat

V rámci této kapitoly je navržen princip nového způsobu generování klíče → polymorfního generování klíče. V rámci klasických způsobů generování klíče, viz kapitola 2.5, se postupuje obvykle následovně. Na vstupu jsou data pro odvození klíče – například heslo, soubor apod., která jsou vstupem pro fixní funkci, která na jejich základě vygeneruje klíč. Pokud se změní data pro odvození klíče, proces výpočtu klíče zůstává vždy stejný.

V rámci článku prezentovaného na konferenci (viz reference autora [A.7]) byl navržen a otestován odlišný přístup. Přístup, který postupuje polymorfně v závislosti na datech, z kterých má být odvozen klíč. Proces výpočtu klíče lze shrnout následovně. Samotná implementace je navržena pro potřeby šifrovacího systému. Tedy pro tvorbu šifrovacího klíče délky 111 bajtů.

Na vstupu jsou data → libovolné délky (pro praktickou implementaci bylo zvoleno minimum 128 bajtů, aby byl zpracován minimálně jeden blok dat). Data mohou být jakýkoliv obsah libovolného souboru.

Data jsou rozdělena do jednotlivých bloků, které jsou zpracovávány samostatně (pro praktickou ukázku po 128 bajtech).

Pokud data nejsou celočíselně dělitelná 128, tak se uloží zbytek dat pro zpracování.

Proces odvození klíče lze rozepsat následovně:

1. Každých 128 bajtů je ještě rozděleno na dvě poloviny
2. Z každé poloviny se vypočítá samostatný hash pomocí hashovací funkce SHA3-512 (Keccak) → délka 64 bajtů.
3. Jelikož máme k dispozici 128 bajtů dat a výstupem je 111 bajtů, tak můžeme využít posledních 17 bajtů „Redundantních dat“ k modifikaci klíče. Tato modifikace je závislá na obsahu těchto 17 bajtů.
4. Z druhého hashe se nyní vezme posledních 17 bajtů a vypočítá se suma hodnot jednotlivých bajtů a případě, že je-li tato hodnota sudá, tak 17 bajtů dat ovlivní první hash. V opačném případě se data zahrnout do druhého hashe.
5. Suma součtu hodnot jednotlivých bajtů posledních bajtů se vydělí 48 a získáme pozici prvního bajtu v rámci hashe, ke kterému bude přičtena hodnota prvního bajtu posledních 17 bajtů.
6. Celý proces se opakuje, pokud máme k dispozici další data o velikosti 128 bajtů pro odvození klíče.
7. Mezi jednotlivých kroky jsou 128 bajtové bloky po jednotlivých bajtech sečteny a modulovány hodnotou 256.
8. Zahrnutí zbývajících dat, pokud data nejsou dělitelná beze zbytku 128. Zpracování zbývajících dat. Proces zpracování zbytku je následující.
9. Zbývající data se také zahashují funkcí SHA3-512 a získáme 64 bajtů dat.
10. Vypočítá suma hodnot bajtů hashe
11. Vypočítá se index prvního bajtu ze 128, ke kterému má být přičten první bajt hashe
12. Provede se postupné sečtení bajtů modulováno 256
13. Klíč je vrácen jako prvních 111 bajtů ze 128 vygenerovaných bajtů.

Podrobný popis algoritmu a analýza byla provedena v rámci článku [A.7]. Nicméně, uvedený princip je pouze ukázková možnost, jak aplikovat myšlenku polymorfnosti na tvorbu a odvození inicializačního vektoru (prvního šifrovacího klíče). Konkrétní implementace byla vytvořena hlavně z důvodu vytvoření ukázkové aplikace jako součást výsledného kryptografického systému TSS. Nejedná se tedy o finální (bezpečnou) a aplikovatelnou verzi do praxe, ale prvotní návrh. Pro aplikovatelnou verzi by bylo nutné provést podrobnou kryptoanalýzu a optimalizaci.

7.4 Parametrizace vlastností systému

Při návrhu šifrovacího systému TSS byly zvoleny vhodné parametry a byl navržen proces parametrizace. Celkově lze parametrizaci rozdělit na dvě části:

- Parametrizace vlastností šifrování – variabilita

    - Speciální parametrizace režimu činnosti PM-DC-LM pro inicializaci hodnot – rozebráno v rámci kapitoly 7.5.
- Parametrizace solení – solení je rozebráno v rámci kapitoly 7.2.

Celkově bylo doposud navrženo 19 parametrů, které do šifrování vstupují a může je uživatel na začátku nastavit a které následně upravují celý proces šifrování. V budoucnu se mohou tyto parametry ještě rozšířit. Některé z parametrů jsou nastaveny pouze na začátku a ovlivňují šifrování a vlastnosti v prvotních fázích šifrování, jiné ovlivňují průběh šifrování v rámci celého procesu.

Parametry jsou navrženy tak, že mají hlavní složku → hodnota parametru, dále mají řídící složku, která slouží k manipulaci a přepočtu parametru pomocí klíče → který bajt klíče bude využitý k přepočtu.

V neposlední řadě jsou zde parametry, které ovlivňují solení, tyto parametry mají obdobně hodnotu parametru a hodnotu řídícího parametru. Většina parametrů má svou funkci na přepočet pomocí klíče. Takže můžeme specifikovat, že návrh jednotlivých parametrů byl v korelaci s následujícím:

- Volba vhodných parametrů (jsou rozepsány v rámci následujících kapitol)
- Způsob určení parametru na začátku
    - Volba na počátku uživatelem – tajemství rozšiřující bezpečnost klíče
- Způsob určení řídící proměnné parametru
    - Volba na počátku uživatelem – tajemství rozšiřující bezpečnost klíče
- Způsob přepočtu parametrů v průběhu šifrování
- Způsob začlenění parametrů do šifrovacího systému

V rámci následujících kapitol jsou popsány parametry, které byly zvoleny, a které byly zahrnuty do výsledného návrhu šifrovacího systému TSS. Řídící parametry budou rozebrány samostatně, protože mají jednotnou přepočtovou funkci. Parametrizace je sice zvolena tak, aby zapadala do kontextu celého šifrovacího systému TSS, ale spousta parametrů by bylo možné využít i stávajících šifrovacích algoritmů, jako je AES. Tyto možnosti a porovnání jsou rozebrány v rámci podkapitol věnovaných jednotlivým navrženým parametrům.

7.4.1 Parametry pro výpočet počtu rund

Parametr slouží pro určení počtů rund v rámci šifrování jednotlivých bloků dat. Každý šifrovaný blok dat má svoji hodnotu počtu rund. Tato hodnota je

odvozena od vstupního nastavení a hodnoty klíče. V rámci počtu rund je vypočítávána i hodnota řídící složky, která se mění v rámci procesu šifrování. Pomáhá určit vhodný bajt aktuálního klíče pro výpočet aktuálního počtu rund pro šifrování bloku dat. Proces určení a práce s parametrem je následující.

Na počátku jak hodnotu defaultního nastavení počtu rund, tak i hodnotu řídícího parametru pro výpočet počtu rund zvolí uživatel → nastaví se jako vstup.

Hodnota počtu rund a řídícího parametru se upraví pomocí hodnoty aktuálního klíče. Platí i pro první i pro aktuální šifrovaný blok dat.

Stanovení počtu rund se provede na začátku pro šifrování prvního bloku dat z nastavení uživatele i ze vstupního klíče → „nasolené“ *IV*. Dále se počet rund a řídící parametr přepočítají před každým dalším šifrování následujícího bloku dat, opět na základě aktuálního počtu rund a hodnoty řídícího parametru a pomocí aktuálního klíče.

Pro praktickou implementaci byly zvoleny následující omezení a nastavení:

- Uživatel si může zvolit hodnotu parametru pro výpočet počtu rund v rozmezí $\langle 0; 255 \rangle$.
- Relevantní hodnoty souvisí s počtem rund, nicméně lze v budoucnu měnit.6
- Uživatel si volí hodnotu řídícího parametru pro výpočet počtu rund v rozmezí $\langle 0; 110 \rangle$.

Hodnota nemůže být větší jako 110 z důvodu, že reprezentuje bajt aktuálního klíče, který bude použitý pro výpočet počtu rund daného bloku.

Počet rund v rámci šifrování je omezen na hodnoty od 1 do 4. Je to z důvodu, aby nedošlo k velmi výraznému nárůstu času šifrování, protože šifrování je nejvíce náročná operace. Praktická implementace a návrh šifrovacího systému ovšem umožňuje, aby bylo možné tuto hranici počtu rund měnit. Například jako parametr od uživatele, který bude zvolen na počátku šifrování.

---

[6] *V rámci aktuální implementace TSS je maximum počtu rund nastaveno na 4. Protože vyšší hodnoty jsou při přepočtu modulovány 4. (přičítá se jedna, 0 rund není povoleno)*

Praktický výpočet počtu rund systému TSS je možné vidět ze zdrojového kódu samotné implementace.

```
def PrepocitejPocetRund(klic, defPromenna, defRidiciPromenna):
    return ((defPromenna + klic[defRidiciPromenna]) % 4) + 1
```

Obrázek 22 - Funkce pro přepočítání počtu rund (vlastní)

Jak lze vidět, tak počet rund souvisí s hodnotou bajtu klíče, která je hodnotou bajtu na parametricky odvozené pozici.

7.4.1.1 Shrnutí a porovnání se stávajícími šifrovacími algoritmy

Prakticky všechny známé a používáme blokové šifry mají jednoznačný počet rund, který je znám na začátku šifrování nebo který je případně zvolen v závislosti na délce klíče. Počet rund se v rámci šifrování nemění. V rámci navrženého šifrovacího systému je nutné pro znalost počtu šifrovacích rund znát následující:

- Hodnotu nastavení parametru počtu rund od uživatele (na začátku)
- Hodnotu nastavení řídícího parametru pro výpočet počtu rund od uživatele (na začátku)
- Klíč
- V rámci solení – sůl

7.4.1.2 Možné využití u stávajících šifer

V rámci šifry AES by bylo například možné využít tuto část parametrizace, aby došlo k manipulaci s počtem rund. Bylo by ovšem nutné použít další klíč pro dopočet potřebných dalších rundovních klíčů. V rámci šifry AES je totiž jasně definováno, že počet rund je závislý na délce klíče a je neměnný. [1]

Jedná se o stanovení počtu rund tak, aby bylo docíleno minimální potřebné bezpečnosti a zároveň, dále aby bylo šifrování dostatečně rychlé. Tedy pro velkou rychlou a zaručení bezpečnosti byla zvolena hodnota 10, kdy pro další varianty klíče je možné využít jeho délky, aby byly odvozeny další rundovní klíče. Modifikace by mohla být následovná:

- Použití dalšího klíče – například klíč délky 64 bitů
- 64 bitů by umožňovalo zvýšit počet rund o 5 – využití AES key-schedule [1]

- Tudíž by šlo využít AES s variabilním počtem rund závislém na klíči a to následovně

7.4.2 Parametry pro výpočet délky šifrovaného bloku

Parametr slouží pro určení délky bloku dat v rámci šifrování jednotlivých bloků dat. Každý šifrovaný blok dat má svoji délku a počet bajtů které budou šifrovány pomocí aktuálního klíče. Tato hodnota je odvozena od vstupního nastavení a hodnoty klíče. V rámci délky bloku pro šifrování je vypočítávána i hodnota řídící složky, která se mění v rámci procesu šifrování a pomáhá určit vhodný bajt aktuálního klíče pro výpočet aktuální délky bloku pro šifrování.

Proces určení a práce s parametrem je následující. Hodnota délky šifrovaného bloku může být maximálně 111, protože to vychází z návrhu šifrovacích funkcí (jádro šifrovacího systému) a délky klíče. Spodní hranice je jeden bajt. To nám umožňuje, aby nebyla potřeba doplňování dat vůči délce bloku. Přepočet a vlastnosti klíče jsou rozebrány v kapitole 7.5.

- Defaultní nastavení je fixní a je rovno délce klíče.
- Na počátku hodnotu řídícího parametru pro výpočet počtu rund zvolí uživatel $\rightarrow$ nastaví se jako vstup.
- Délka bloku a řídícího parametru se upraví pomocí hodnoty aktuálního klíče. Platí i pro první i pro aktuální šifrovaný blok dat.
- Výpočet délky bloku dat je ovlivněn dalším parametrem – variabilita délky bloku dat (rozebráno v následující kapitole)

Vypočítání délky prvního šifrovaného bloku dat se provede na začátku z nastavení poskytnutých uživatelem a ze vstupního klíče. Velikosti následujících šifrovaných bloků dat a řídících parametrů se přepočítají před každým dalším šifrování následujícího bloku dat, opět na základě aktuální velikosti bloku a hodnoty řídícího parametru a pomocí aktuálního klíče (zde ještě na základě parametru variability délky bloku).

Pro praktickou implementaci byly zvoleny následující omezení a nastavení:

Uživatel si volí hodnotu řídícího parametru pro výpočet délky bloku v rozmezí $\langle 0; 110 \rangle$.

Hodnota řídícího parametru nemůže být větší jako 110 z důvodu, že reprezentuje bajt aktuálního klíče, který bude použitý pro výpočet počtu rund daného bloku.

Výsledná hodnota délky bloku se odvíjí od hodnoty variability délky bloku a délka šifrovaného bloku může být hodnota na intervalu $\langle 91; 111 \rangle$.

```
def PrepocitejDelkuBloku(ridiciPromenna, variabilitaBloku, klic):
    return 111 - (klic[ridiciPromenna] % variabilitaBloku)
```

Obrázek 23 - Funkce pro přepočet délky bloku pro šifrování (vlastní)

Lze říci, že délka bloku souvisí se vstupními parametry, tak i všemi aktuálními stavy klíčů, otevřených dat, šifrových dat a náhodných solí.

7.4.2.1 Parametr pro výpočet variability délky bloku

Jelikož se jedná o parametr, který přímo souvisí s délkou bloku, je vhodné jej rozepsat zde. Dá se říci, že je to hodnota udávající rozdíl mezi aktuální délkou bloku a defaultní délkou bloku, která se rovná 111. Variabilita délky bloku je řešena obdobně jako počet rund. Je určena na základě hodnot od uživatele (vstupní hodnota) a na základě řídící proměnné, která je také určena na začátku před šifrováním. Dála je hodnota přepočítána pro šifrování a výpočet délky prvního bloku a následně přepočítávána pro každý následující šifrovaný blok dat.

```
def PrepocitejVariabilituBloku(promenna, ridiciPromenna, klic):
    return ((promenna + klic[ridiciPromenna]) % 10) + 11
```

Obrázek 24 - Funkce pro přepočet variability délky bloku (vlastní)

Uživatel může volit řídící parametr na intervalu $\langle 0; 110 \rangle$, opět omezený dle délky klíče. Parametr pro určení délky lze volit na rozmezí 0 až 255, kdy k ovlivnění průběhu dochází v rámci hodnot 0 až 9, ale je zde prostor pro případné budoucí změny. Nicméně zde teoreticky zatím po hodnoty 110, aby nedošlo k šifrování nulových bloků. Výsledná variabilita může nabývat hodnot na intervalu $\langle 0; 20 \rangle$.

7.4.2.2 Shrnutí a porovnání se stávajícími šifrovacími algoritmy

Prakticky všechny známé a používáme blokové šifry mají jednoznačnou délku šifrovaných bloků, který je znám na začátku šifrování nebo který je případně zvolen v závislosti na délce klíče. Délka se pak během šifrování nemění. Při využití šifrovacího systému TSS lze pro určení délky šifrovaného bloku vycházet na základě:

- Defaultní hodnoty délky bloku rovné 111.
- Nastavení řídící proměnné pro určení délky bloku od uživatele (na počátku)
- Nastavení vstupní hodnoty variability délky bloku dat od uživatele (na začátku)

- Hodnotu nastavení řídícího parametru pro výpočet variability délky bloku od uživatele (na začátku)
- Klíč
- V rámci solení – sůl

7.4.2.3 Možné využití u stávajících šifer

Pro využití u stávajících šifer by bylo nasazení obtížné, jelikož mají fixní délku šifrovaných bloků, které jsou ruku v ruce s šifrovacím jádrem šifer. Zde by bylo zapotřebí přepracování značné části šifry. V rámci šifrovacího systému TSS byly všechny součásti přepracovány tak, aby to bylo možné.

7.4.3 Parametry pro stanovení proměnné N

Parametr proměnná N je speciální parametr, který slouží k parametrizaci a změně procesu výpočtu šifrovacího klíče v rámci režimů činnosti. Podobně jako výše uvedené parametry je jeho hodnota odvozena od vstupního nastavení uživatelem na začátku šifrování a dále v rámci šifrování přepočítáván na základě aktuálního nastavení a parametrů. Tato proměnná v rámci výpočtu klíče pro šifrování následujícího bloku slouží jako posun hodnot vypočtených bajtů. Parametr může nabývat hodnot v rámci intervalu 0 až 255. Hodnoty vyšší jako 255 by byly ořezány velikostí bajtů (modulace hodnotou 256). Při určení parametru se postupuje následovně:

- Uživatel nastaví hodnotu proměnné N
- Uživatel nastaví řídící parametr pro určení hodnoty parametru N
- Na základě klíče spolu s řídícím parametrem je vypočtena hodnota proměnné N. Ta je připravena pro přepočet klíče pro následující blok dat.
- Po zašifrování bloku dat dojde k přepočítání hodnoty proměnné N, aby byl připraven na výpočet dalšího klíče

Pro praktickou implementaci byly zvoleny následující omezení. Základní hodnota může být uživatelem zvolena jako číslo na intervalu <0, 255>. Pro přepočet proměnné N je využita stejná přepočetní funkce jako pro proměnnou X. Následující obrázek zachycuje její implementaci.

```
def PrepocitejPromennou(klic, defPromenna, defRidiciPromenna):

    return (defPromenna + klic[defRidiciPromenna]) % 256
```

Obrázek 25 - Funkce pro přepočet proměnnýc N a X (vlastní)

7.4.3.1 Shrnutí a porovnání se stávajícími šifrovacími algoritmy

Prakticky všechny známé a používáme blokové šifry mají předem definovaný klíčovací rozvrh (key-schedule) a nelze s ním před začátkem šifrování

nakládat. V práci bylo rozebráno, jaké hodnoty a parametry ovlivňují nebo vstupují do výpočtu jednotlivých bajtů nového klíče. U většiny stávajících šifer probíhá také výpočet či odvození nových klíčů. Pokud se ale jedná a jejich parametrizaci, tak ve výsledku se jedná a lineární parametr, který je pravidelně inkrementován. V lepším případě se jedná o *nonce* (šum) či zmíněný tweak, které je ovšem nutné přenášet spolu s šifrovým textem či jinak předat adresátovi před dešifrováním. V porovnání se stávajícími je zde parametr volen na základě uživatelovi volby – tajemství. Dále je upraven na základě klíče a v případě solení i na základě náhodných hodnot:

- Nastavení řídící proměnné pro určení proměnné N od uživatele (na počátku)
- Nastavení vstupní hodnoty proměnné N od uživatele (na začátku)
- Klíč
- V rámci solení – sůl

7.4.3.2 Možné využití u stávajících šifer

Pro využití u stávajících šifer by to nebylo snadné, jelikož mají stabilní či specificky navržený key-schedule. V rámci šifrovacího systému TSS byly všechny součásti přepracovány tak, aby to bylo možné a v celkovém souladu s polymorfností. Viz režimy činnosti v rámci kapitoly 7.5.

7.4.4 Parametry využívané pro solení

I když je proces solení podrobně rozebrán v rámci kapitoly 7.2, tak si zde uvedeme základní parametry, které jsou v rámci procesu solení zvoleny uživatelem a jak se parametry mění v závislosti na *IV* nebo následně na *IV* po „nasolení“.

Pro proces solení jsou na vstupu následující uživatelské parametry → podobně jako ostatní parametry si je volí uživatel jako tajné před procesem šifrování.

- Defaultní hodnota délky „předsolení“
  - Označme jako dl_p
- Defaultní hodnota řídící proměnné pro délku „předsolení“
  - Označme jako $dř_p$
- Defaultní hodnota délky „nasolení“
  - Označme jako dl_n
- Defaultní hodnota řídící proměnné pro délku „nasolení“
  - Označme jako $dř_n$
- Defaultní hodnota délky „zasolení“

    - Označme jako dl_z
- Defaultní hodnota řídící proměnné pro délku „zasolení“
    - Označme jako $dř_z$
- Defaultní hodnotu indexu pro výpočet pozice soli pro „nasolení“
    - Označme jako dip_n
- Defaultní hodnotu řídící proměnné pro výpočet indexu pozice soli pro „nasolení“
    - Označme jako $dřip_n$

Z výše uvedených dl_p a $dř_p$ slouží pro určení variabilní délky „předsolení“ → hodnota celého čísla na intervalu $\langle 0; 255 \rangle$ (viz kapitola 7.2.2). Dílčí řídící proměnná slouží pro volbu bajtu klíče, který bude použitý pro výslednou hodnotu variabilní části hodnoty délky „předsolení“, a tudíž může nabývat hodnoty celého čísla na intervalu $\langle 0; 110 \rangle$, protože délka klíče je 111 bajtů. Stejně je to s hodnotami dl_n, $dř_n$, dl_z a $dř_z$ pro „nasolení“ a „zasolení“.

Odlišně je to s hodnotami parametrů pro výpočet indexu → dip_n a $dřip_n$, které jsou využité pro výpočet pozice soli v rámci šifrových dat. Výpočet pozice soli v šifrových datech je popsaný v rámci kapitole věnované solení → kapitola 7.2.5. Jak je patrné z textu uvedené kapitoly, tak se využívá 11 bajtů *IV* k určení pozice soli ve výsledných šifrových datech, proto defaultní hodnota indexu pro výpočet pozice soli může nabývat hodnot celých čísel na intervalu $\langle 0; 109 \rangle$. Defaultní hodnotu řídící proměnné pro výpočet indexu pozice soli je možné volit stejně jako u všech řídících proměnný → celé číslo na intervalu $\langle 0; 110 \rangle$.

Pro potřeby následujícího textu si označme inicializační vektor za *IV* a inicializační vektor po „nasolení“ jako IV_n. Hodnoty délky „nasolení“ → dl_n a hodnota indexu pro výpočet pozice soli → dip_n jsou vypočteny na základě dat *IV* a hodnoty dl_p a dl_z jsou vypočteny na základě dat IV_n. Pro výpočty jsou využité následující rovnice.*

$$l_p = 111 + \left(\left(dl_p + IV_n[dř_p] \right) mod\ 256 \right)$$

$$l_n = 111 + \left((dl_n + IV[dř_n]) mod\ 256 \right)$$

* *Hranaté závorky v rámci IV nebo IV_n reprezentují pozici bajtu.*

$$l_z = 111 + \left((dl_z + (IV_n[111 - d\check{r}_z])\ mod\ 111\ )\ mod\ 256\right)$$

$$ip_n = \left((dip_n + IV[d\check{r}ip_n])\ mod\ 100\right)$$

Rovnice pro výpočet délky „předsolení“ a „zasolení“ byly zvoleny záměrně odlišné, aby docházelo k rozdílnému chování výsledného „předsolení“ a „zasolení“. Rovnice pro výpočet délky „předsolení“ a „nasolení“ mají stejný tvar, ale každá vychází z jiných dat *IV* → délka „předsolení“ na základě „nasoleného“ *IV* a délka „nasolení“ z *IV* bez „nasolení“.

7.4.5 Parametry pro stanovení hodnoty proměnné X

Stanovení hodnoty proměnné X dochází na začátku před šifrováním prvního bloku. Tato hodnota je ovlivněna uživatelskými tajnými vstupními parametry → defaultní hodnota proměnné X a defaultní hodnota řídící proměnná pro nastavení parametru X a dále je přepočtena za využití „nasoleného“ *IV*.

Hodnota zvolená uživatelem jako defaultní hodnota proměnné X je celé číslo na intervalu $\langle 0;\ 255\rangle$. Defaultní hodnotu řídící proměnné pro výpočet proměnné X je možné volit jako hodnotu celého čísla na intervalu $\langle 0;\ 110\rangle$.

Proměnná X slouží jako parametr určující index bajtu klíče použitého pro přepočet řídících proměnných, které následně slouží pro přípravu parametrů → počet rund, variabilitu délky šifrovaného bloku a s tím souvisejícímu následnému přepočtu délky šifrovaného bloku a pro přepočet proměnné N (využívaná v rámci režimů činnosti a odvození šifrovacích klíčů). Proměnná X je po dokončení šifrování bloku dat ještě iterována hodnotou počtu použitých rund pro šifrování.

Přepočet proměnné X na začátku lze zachytit zdrojovým kódem uvedeného obrázkem číslo 23.

7.4.6 Funkce pro přepočet dílčích řídících parametrů

Jak lze vidět z předchozích kapitol, tak jednotlivé parametry mají vždy svůj řídící parametr, který slouží v přepočtu zejména pro určení indexu bajtu klíče, který ovlivní samotný výpočet a výslednou hodnotu parametrů.

Jedná se o následující parametry:

- Parametr pro výpočet počtu rund
- Parametr pro stanovení hodnoty proměnné N
- Parametr pro výpočet délky šifrovaného bloku
- Parametr pro stanovení hodnoty variability délky šifrovaných bloků
- Parametr pro výpočet hodnoty proměnné X

- Parametr pro výpočet hodnoty potřebné pro režim činnosti PM-DC-LM

Kdy z výčtu výše první čtyři a červeně zvýrazněné jsou řídící parametry, které jsou přepočítané před šifrováním prvního bloku pomocí „nasoleného“ *IV* a pak i vždycky po šifrování bloku dat jako re-parametrizace před šifrování bloku následujícího. Zbylé dva parametry nemají přepočetní funkci a slouží pouze ve formě v jaké jsou na vstupu jako tajné uživatelské parametry. K přepočítání řídících parametrů dochází opět na základě aktuálního klíče. Toto přepočítání lze reprezentovat následující implementací zdrojového kódu.

```
def PrepocitejRidiciPromennou(ridiciPromenna, promennaX, klic, posun):
    return (ridiciPromenna + klic[(promennaX + posun) % 111]) % 111
```

Obrázek 26 - Funkce pro přepočet řídících proměnných parametrů (vlastní)

Posun je hodnota, která upravuje hodnotu indexu bajtu klíče pro přepočet. Pro každou ze čtyř výše uvedených proměnných je posun jiný → hodnoty 1, 2, 3 a 4. Ve výsledku jsou proměnné přepočteny na základě 4 po sobě jdoucích bajtů klíče. Pozice indexů bajtů klíče je parametricky vyjádřena proměnnou X.

7.4.7 Parametry pro režim činnosti PM-DC-LM

Jak bylo uvedeno v úvodu kapitoly věnované parametrizaci, tak kromě parametrů potřebných pro variabilitu systému a parametrů pro solení byly navrženy dva další parametry. Tyto parametry slouží pouze pro účely použití režimu činnosti PM-DC-LM (varianta 5, viz kapitola 7.5.3.5). Jedná se o parametry, které slouží k inicializaci CPRNG režimu PM-DC-LM. Parametry jsou opět ve formě dvojice a jsou tajným uživatelským vstupem v rámci nastavení → defaultní hodnota parametru a defaultní hodnota řídícího parametru. Inicializace, a tudíž i využití parametrů dochází pouze jednou, před začátkem šifrování prvního bloku otevřených dat. Hodnota proměnné pro inicializaci CPRNG je celé číslo na intervalu $\langle 9;\ 14\rangle$ a slouží jako počet prvotních generování CPRNG → výsledkem je „rozběhnutí“ systému a dosažení chaotického chování logistických map.

Rovnice použitá pro prvotní nastavení a výpočet počtu běhu CPRNG lze vidět z následující implementace.

```
def PrepocitejPocetBehu(klic, defPromenna, defRidiciPromenna):
    return ((defPromenna + klic[defRidiciPromenna]) % 6) + 9
```

Obrázek 27 - Funkce pro výpočet počtu běhu CPRNG - inicializace (vlastní)

7.4.8 Shrnutí a možnosti do budoucna

Jak bylo uvedeno v úvodu kapitoly, tak doposud bylo vytvořeno 19 parametrů, které lze v rámci uživatelského nastavení před započetím šifrování nastavit. Jedná se o parametry pro změnu variability šifrování (9), parametry pro nastavení solení (8) a parametry speciálně pro potřeby režimu činnosti PM-DC-LM (2). Celková režie parametrizace a přepočítávání parametrů je jeden z důvodů, který velmi pozitivně ovlivňuje bezpečnost systému TSS a inovativně rozšiřuje problematiky symetrické kryptografie, ale bohužel je i jeden z důvodů, který má negativní dopad na dobu šifrování.

Nicméně tyto parametry lze dle potřeby měnit → ubírat i přidávat a parametrizovat další části/vlastnosti systému. Pro potřeby vylepšení systému TSS je v plánu přidání dalších parametrů, například [7]:

- Parametricky odvozená pozice hodnot *d* a *g* v CPRNG režimu činnosti PM-DC-LM (aktuálně jsou fixní)
- Parametrické vyjádření pozice soli pro „nasolení“ v šifrovém textu
- Parametry pro šifrovací jádro → indexování či omezení přičítání klíče

7.5 Návrh správy klíčů a tvorba režimu činnosti blokové šifry

V této kapitole jsou uvedeny výsledky výzkumu, který byl realizován v rámci doktorského studia. Výsledky byly i z velké části prezentované a publikované v rámci vědeckých konferencí. Dosavadní publikované výsledky se týkají zejména návrhu polymorfního režimu činnosti, který byl nazván PM-DC-LM a může být zobecněn jako skupina režimů činnosti PM. Princip je založen na souběžném generování pseudo-náhodných hodnot generátor založeném na deterministickém chaosu logistických map.

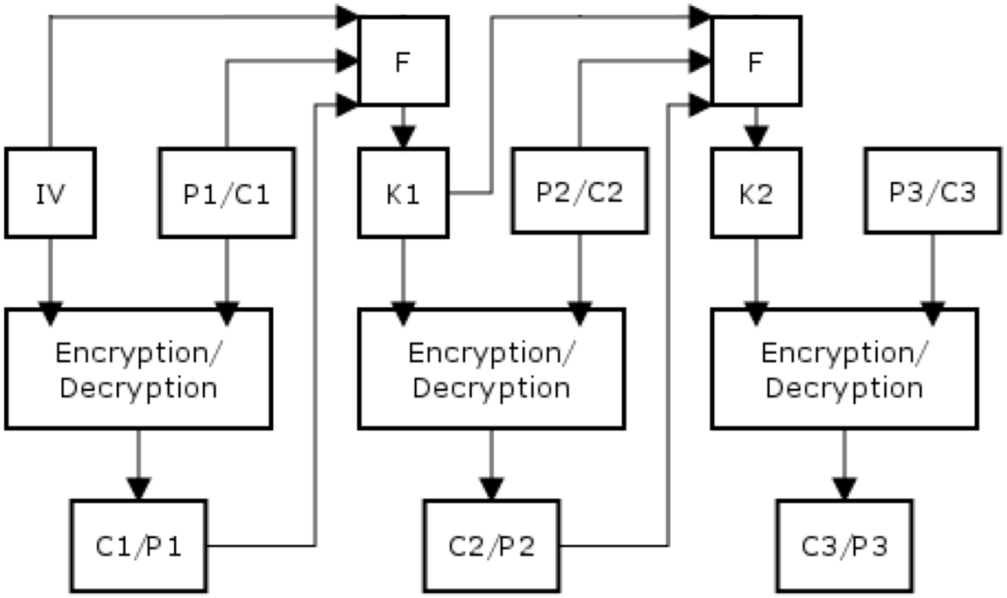
Všechny návrhy (varianty), které byly vytvořeny v rámci studia lze označit za experimentální aplikaci v souladu s myšlenkou polymorfnosti. Nejedná se o bezpečnou plně v praxi využitelnou variantu a jsou podkladem k dalšímu výzkumu podrobnější analýze.

7.5.1 Skupina polymorfních režimů činnosti PM

[7] *Určitě se nejedná o kompletní výčet vlastností, které je možné parametrizovat a řídit na základě klíče.*

V porovnání s aktuálně používanými režimy činnosti, které využívají pouze jednu složku z předchozích dat (otevřená data nebo šifrová data) k modifikaci aktuálních dat (opět otevřená nebo šifrová data), polymorfní režim činnosti mění klíč, který byl použit k zašifrování bloku dat na jiný, který bude použit pro šifrování dat následující bloku.

Za takovýto režim činnosti může být označen i režim činnosti z diplomové práce. Zmíněný režim k odvození jednoho bajtu dat využívá kombinaci bajtů předchozího otevřeného textu, aktuálního šifrového textu a posledního využitého klíče k šifrování. Na základě početní kombinace bajtu každé složky je odvozená rovnice pro výpočet jednoho bajtu nového klíče. Princip lze zobecnit a je znázorněný následujícím diagramem.



Obrázek 28 – Diagram režimu činnosti ze skupiny PM (vlastní)

- *IV* – označuje inicializační vektor („nasolené“ *IV* v rámci TSS)
- P_i – označuje otevřený blok dat
- C_i – označuje šifrový blok dat
- K_i – označuje klíč pro šifrování/dešifrování bloku dat
- F – Funkce, která na základě složek – parametr N (viz kapitola 7.4.3), P_i, C_i a K_i odvodí klíč K_{i+1}

Diagram zachycuje obecnou strukturu polymorfního režimu činnosti. Na základě uvedeného diagramu je možné měnit funkci F a tím vytvořit či optimalizovat režim činnosti dle potřeby. Uveďme si tři základní návrhy

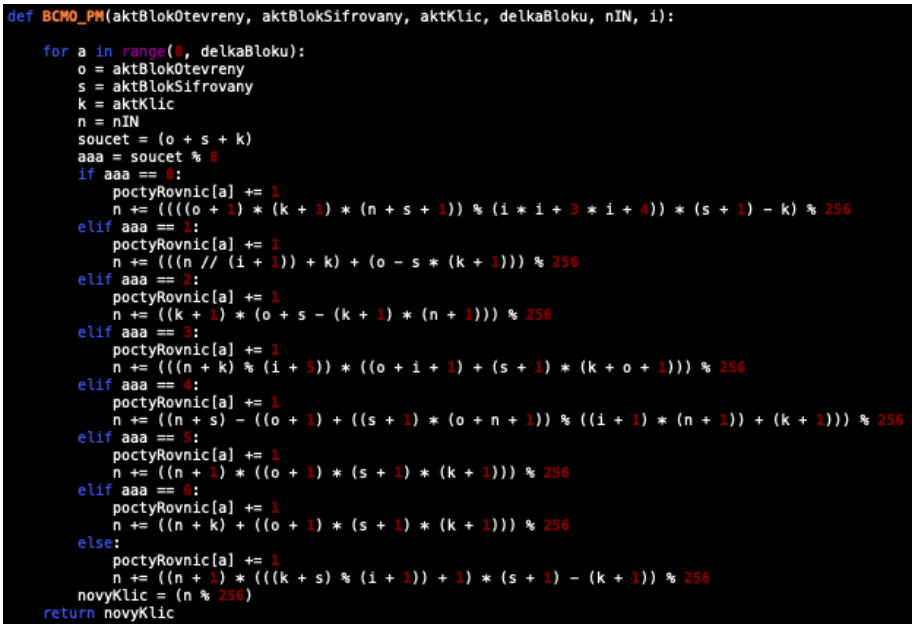
polymorfního režimu činnosti → varianta 1, varianta 2 a varianta 3. Popíšeme si ještě speciální modifikaci jako variantu 4 s označením „PM-PolyUltra“. První je přímou implementací v diplomové práci a druhá varianta je jeho vylepšenou verzí v rámci náplně disertační práce → vylepšení pravděpodobnostní distribuce volby rovnic. Ukázky a testy jsou provedeny implementací v programovacím jazyce Python 3.x.

```
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*i + 3*i + 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 29 – Zdrojový kód režimu činnosti PM před optimalizací – varianta 1 (vlastní)

Výše uvedená aplikace a samotný návrh se skládal z několika kroků. Určující složka je označena jako hodnota *n* → reprezentuje hodnotu nového bajtu klíče. Parametr na základě, kterého se volí rovnice v rámci funkce F je vypočtena jako součin z aktuálních hodnot bajtu předchozího klíče, předchozích otevřených dat a předchozích šifrových dat. Hodnoty jsou patřičně zvýšeny o jedna, aby nedocházelo ke zvýšenému výskytu nul. Z tohoto součinu byla zvolena rovnice výpočtu bajtu následujícího klíče. V začátcích studia dané problematiky byla provedena analýza, která ukázala, že rozložení volby rovnice bylo nesouměrné. Některé rovnice byly voleny s vyšší a některé s menší pravděpodobností v porovnání s pravděpodobností jedna ku osmi (viz kapitola 7.5.3). Na základě tohoto negativního faktoru byla provedena optimalizace, která algoritmus upravuje následovně



```
def BCMO_PM(aktBlokOtevreny, aktBlokSifrovany, aktKlic, delkaBloku, nIN, i):

    for a in range(0, delkaBloku):
        o = aktBlokOtevreny
        s = aktBlokSifrovany
        k = aktKlic
        n = nIN
        soucet = (o + s + k)
        aaa = soucet % 8
        if aaa == 0:
            poctyRovnic[a] += 1
            n += ((((o + 1) * (k + 1) * (n + s + 1)) % (i * i + 3 * i + 4)) * (s + 1) - k) % 256
        elif aaa == 1:
            poctyRovnic[a] += 1
            n += (((n // (i + 1)) + k) + (o - s * (k + 1))) % 256
        elif aaa == 2:
            poctyRovnic[a] += 1
            n += ((k + 1) * (o + s - (k + 1) * (n + 1))) % 256
        elif aaa == 3:
            poctyRovnic[a] += 1
            n += (((n + k) % (i + 5)) * ((o + i + 1) + (s + 1) * (k + o + 1))) % 256
        elif aaa == 4:
            poctyRovnic[a] += 1
            n += ((n + s) - ((o + 1) + ((s + 1) * (o + n + 1)) % ((i + 1) * (n + 1)) + (k + 1))) % 256
        elif aaa == 5:
            poctyRovnic[a] += 1
            n += ((n + 1) * ((o + 1) * (s + 1) * (k + 1))) % 256
        elif aaa == 6:
            poctyRovnic[a] += 1
            n += ((n + k) + ((o + 1) * (s + 1) * (k + 1))) % 256
        else:
            poctyRovnic[a] += 1
            n += ((n + 1) * (((k + s) % (i + 1)) + 1) * (s + 1) - (k + 1)) % 256
        novyKlic = (n % 256)
    return novyKlic
```

Obrázek 30 – Zdrojový kód režimu činnosti PM po optimalizaci – varianta 2 (vlastní)

Na zdrojovém kódu si můžeme povšimnout následujících modifikací.

- Hodnota *nIN* je volena variabilně (viz kapitola 7.4.3) – konstanta modifikující hodnotu bajtu klíče v závislosti na nastavení
- Součin byl převeden na součet, který je opět modulován hodnotou 8, tudíž výběr z 8 rovnic.

Optimalizace, jak bylo zmíněno výše, byla provedena zejména z důvodu přiblížení se ideálnímu rozložení možných výsledků v závislosti na možných vstupech. To znamená, že výběr rovnice pro výpočet bajtu následujícího klíče → jedna ku osmi, tak i pravděpodobnostní rozložení výsledných hodnot bajtů klíče by mělo být co nejvíce rovnoměrné → jedna ku dvě stě padesáti šesti. V tomto případě jsou určujícími vstupy hodnoty:

- $o \rightarrow$ hodnota bajtu přechozích otevřených dat
- $s \rightarrow$ hodnota bajtu posledních šifrových dat
- $k \rightarrow$ hodnota bajtu aktuálního klíče

Každá z výše uvedených hodnot může být celé číslo na intervalu $\langle 0; 255 \rangle$. Proto lze říct, že máme $256 + 256 + 256 = 768$ různých kombinací vstupů na základě kterých bude zvolena jedna z osmi rovnic pro výpočet bajtu následujícího klíče. V ideálním případě by měla být volba rovnice rozprostřena tak, aby na z každých osmi různých vstupních nastavení parametrů byla zvolena právě jedna rovnice. Tedy, aby pro každou z kombinací o, s a k, kterých je celkově $256^3 = 16777216$ byla každá použitá právě 2097152 krát. Varianty 2 i 3 tohoto

rozložení dosahují a testování je v rámci práce ukázané (viz kapitoly 7.5.3.2 a 7.5.3.3). Obecně řečeno by mělo platit, že jsou rovnice využity stejnoměrně.

V případě, pokud bychom chtěli měnit počet rovnic, tak by se mělo jednat o počet, který bude celočíselně beze zbytku dělit počet kombinací určujících hodnot. V našem případě součet hodnot *o*, *s* a *k* → 768 je beze zbytku dělitelné 8. Pak můžeme říct, že bude rovnice volena se stejnou pravděpodobností. Pokud bychom v rámci variant 2 a 3 chtěli změnit počet rovnic jednalo by se o následující počet rovnic na množině

$$\{1, 2, 3, 4, 6, 8, 12, 16, 24, 32, 48, 64, 96, 128, 192, 256, 384, 768\}$$

Při použití vypsaných počtu rovnic bude využití rovnic ideálně distribuované a rovnoměrně rozložené. To vše za předpokladu, že jsou vstupní hodnoty určující rovnici rovnoměrně distribuovány.[*]

V rámci výzkumu došlo ještě k jedné změně tvaru režimu PM. Zejména z důvodu optimalizace výkonu a pokusu změnit rovnice → modifikace tvaru rovnic pro výpočet bajtu následujícího klíče. Výsledkem je varianta 3. To vše za zachování ideální pravděpodobnostního rozložení volby rovnice, kdy počet rovnic zůstává 8 a volba je určena stejným činitel součtu hodnot *o*, *s* a *k*. Zdrojový kód varianty 3 je zachycen následujícím obrázkem a výsledky testů jsou podrobněji rozepsané v následující podkapitole.

[*] *V případě otevřených dat a generovaném IV na základě uživatelských dat by to nebylo možné, ale v rámci implementace TSS jsou jak vstupní data „předsolená“, tak i IV „nasolený“ pomocí solí → viz kapitola 7.2.*

```
novyKlic = bytearray(aktKlic)
for i in range(0, delkaBloku):
    o = aktBlokOtevreny[i]
    s = aktBlokSifrovany[i]
    k = aktKlic[i]
    n = nIN
    soucet = (o + s + k)
    aaa = soucet % 8
    if aaa == 0:
        n += (o + s) % 256
    elif aaa == 1:
        n += (o + k) % 256
    elif aaa == 2:
        n += (o + n) % 256
    elif aaa == 3:
        n += (o - s) % 256
    elif aaa == 4:
        n += (s + k) % 256
    elif aaa == 5:
        n += (o + s + k) % 256
    elif aaa == 6:
        n += (o + s + k + nIN) % 256
    else:
        n += (2*k + s - o) % 256
    novyKlic[i] = ((n + i) % 256)
return bytes(novyKlic)
```

Obrázek 31 - Zdrojový kód režimu PM (varianta 3) (vlastní)

V souvislosti s optimalizací rozložení výstupních hodnot bajtu nového klíče pro šifrování následujícího bloku dat vznikla dílčí varianta skupiny PM, která má kódové označení PM-PolyUltra (varianta 4). V rámci této verze lze říci, že rovnice jako takové byly převedeny pouze v jednu modulaci za využití všech vstupních parametrů o, s, k a n (parametr n – kapitola 7.5.3.4). Zdrojový kód je vidět na následujícím obrázku.

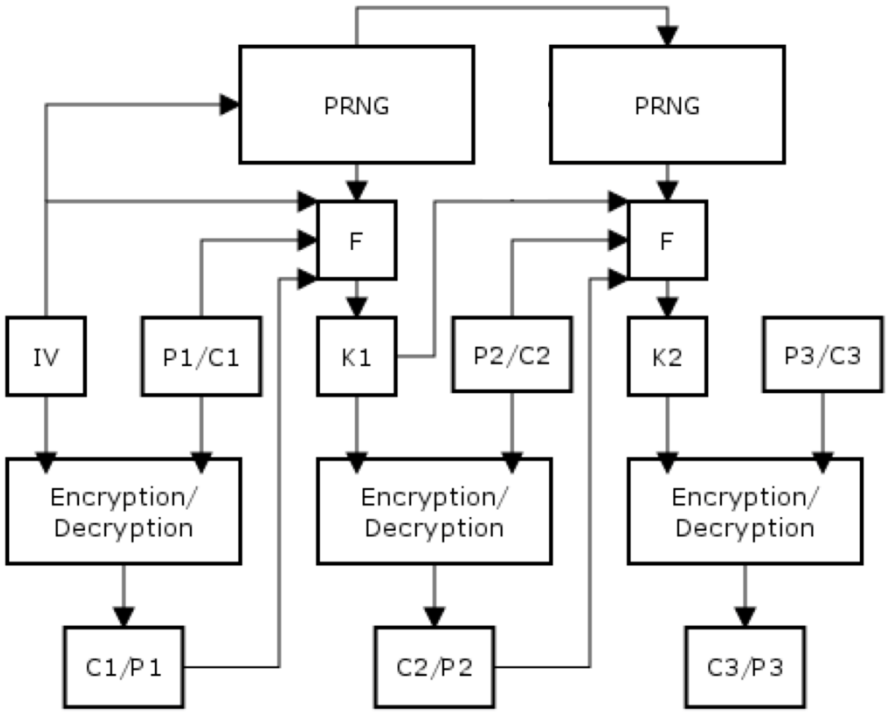
```
novyKlic = bytearray(aktKlic)
for i in range(0, delkaBloku):
    o = aktBlokOtevreny[i]
    s = aktBlokSifrovany[i]
    k = aktKlic[i]
    n = nIN
    p = (o + s + k + n) % 256
    novyKlic[i] = p % 256
return bytes(novyKlic)
```

Obrázek 32 - Zdrojový kód režimu PM-PolyUltra (varianta 4) (vlastní)

Varianta 4 – PolyUltra eliminuje možnost výběru rovnice, ale na druhou stranu poskytuje optimální distribuci pravděpodobností vrácené hodnoty výstupního jednoho bajtu klíče (viz kapitola 7.5.3.4).

7.5.2 Polymorfní režim činnosti PM-DC-LM

V rámci výzkumu byl navržen režim činnosti nazvaný PM-DC-LM, jedná se o modifikaci polymorfního režimu činnosti z kapitoly 7.5.1. Můžeme jej zařadit do skupiny polymorfních režimů činnosti, proto první část zkratky je složena z písmen PM (Polymorphous mode). Jedná se v pořadí o pátou variantu ze skupiny PM. Režim činnosti funguje na principu odvození následujícího klíče pro šifrování, a to na základě bajtů složek otevřených dat, šifrových dat a šifrovacího klíče, jak tomu bylo v obecném schématu, jen je navíc rozšířen o pseudo-náhodný generátor čísel, který vnáší do odvození další složku. V tomto konkrétním příkladu se jedná o chaotický pseudo-náhodný generátor čísel založený na principu deterministickém chaosu logistických map (význam zbývajících zkratek → Deterministic Chaos – Logistic Map). Princip a úpravu zachycuje následující schématický obrázek číslo 31.



Obrázek 33 – Diagram režimu činnosti PM-DC-LM (vlastní)

Z výše uvedeného diagramu vyplývá, že byl přidán další řídící parametr funkce F, a to výstup náhodného generátoru. V rámci výzkumu bylo nutné řešit způsob odvození startovního bodu pro náhodný generátor na základě *IV* (v rámci systému TSS na základě „nasoleného“ *IV*). Dále byl vhodně modifikován obsah funkce F. Celý výzkum včetně výsledků dokumentují publikace [A.1 až A.6] ze seznamů publikační činnosti autora na konci této práce. V rámci těchto publikací došlo k několika modifikacím, analýzám a optimalizacím. Výsledný režim PM-

DC-LM byl následně mírně modifikován, a tudíž i zdrojový kód, viz následující obrázek.

```
def IVtoR(IV, p):

    IVs = ''
    for a in range(0, p):
        s = 0
        for i in range(0, len(IV) // p):
            s = (s + int(IV[a + (i*p)])) % 10
        IVs += str(s)
    k = len(IV) % p
    if k != 0:
        rem = '' + (p - k)*'0' + IV[len(IV) - k:]
        l = list(IVs)
        for i in range(0, p):
            l[i] = str((int(l[i]) + int(rem[i])) % 10)
        IVs = ''
        for i in range(0, len(l)):
            IVs += l[i]
    r = 3.9 + int(IVs)*(10**( - p - 1))
    x = int(IVs)*(10**( - p - 1)) * 10
    return [r, x]
```

Obrázek 34 – Zdrojový kód pro převod IV na hodnotu r pro režim činnosti PM-DC-LM (vlastní)

Obrázek znázorňuje zdrojový kód způsobu odvození hodnoty *r*, použité k nastartování chaotického generátoru pseudo-náhodných čísel na základě *IV*.

V následujícím obrázku můžeme vidět zdrojový kód samotného pseudo-náhodného generátoru založeném na funkci logistických map z deterministického chaosu.

```
def NextFromCPRNG(x, r):

    return x*r*(1 - x)
```

Obrázek 35 – Zdrojový kód CPRNG režimu činnosti PM-DC-LM (vlastní)

Níže uvedený obrázek je zobrazením zdrojového kódu modifikované verze funkce F tak, aby bylo možné aplikovat výstup generátoru, který je dvojicí čísel *d* a *g*, kdy *d* je předposlední číslice hodnoty *x* a *g* je číslo složené z číslici na pozicích 5, 4 a 3 zprava. Hlavní řídícím parametrem funkce F je číslo *d* na nabývající hodnot na intervalu $\langle 0; 9 \rangle$, tudíž nyní je rovnic pro výpočet bajtu následujícího klíče 10.

```
novyKlic = bytearray(aktKlic)
for i in range(0, delkaBloku):
    o = aktBlokOtevreny[i]
    s = aktBlokSifrovany[i]
    k = aktKlic[i]
    strX = str(x)
    d = int(strX[-2])
    g = int(strX[-5] + strX[-4] + strX[-3])
    n = nIN
    if d == 1:
        n += (o + s + d + g) % 256
    elif d == 2:
        n += (s + d + g) % 256
    elif d == 3:
        n += (k + d + g) % 256
    elif d == 4:
        n += (2*o - s + d + g) % 256
    elif d == 5:
        n += (2*s + d + g) % 256
    elif d == 6:
        n += (2*k + d + g) % 256
    elif d == 7:
        n += (3*o + 2*s + d + g) % 256
    elif d == 8:
        n += (3*s + d + g) % 256
    elif d == 9:
        n += (3*k + d + g) % 256
    else:
        n += (4*o - 3*s + d + g) % 256
    novyKlic[i] = (n % 256)
    x = NextFromCPRNG(x, r)
return [bytes(novyKlic), x]
```

Obrázek 36 – Zdrojový kód režimu činnosti PM-DC-LM – varianta 5 (vlastní)

Z analýzy výsledků v rámci výzkumu režimu činnosti PM-DC-LM vyplynulo, že bude potřeba vyspecifikovat a eliminovat některé hodnoty *r*, protože nevedou ke generování čísel s vysokou hodnotou entropie (dosud nepublikováno). Dále lze říci, že pozice čísel *d* a *g* zprava nehraje významnou roli na velikost entropie a pravděpodobnostní rozložení (viz [A.9]).

Nicméně lze vypozorovat, že v rámci odvozování hodnot *r* a *x* dochází ke kolizím a rozložení hodnot v závislosti na různých kombinacích *IV* není rovnoměrné. Z toho důvodu by bylo mnohem lepší, aby pro každé vlastní *IV* byla zvolena právě jedna hodnota *r*, a právě jedna hodnota *x*. V souvislosti s tímto problémem bylo vytvořeno protiopatření ve formě aplikace typu Decimal, který umožňuje počítání s velkými čísly. Vylepšená varianta počítá s hodnotou *IV* nikoliv jako s binárním číslem, které je zpracované po blocích, ale nyní jako s jedním velkým celým decimálním číslem na intervalu $\langle 0;\ 2^{888}\rangle$. Maximální hodnota může být číslo →

20636505122486923685638272848301429942142473673285996958123465196354449318622064823219424058111608902135718554424106589018841701543073653798849178846208577222983854843711136100341074909235407888

536337590979769995470370323551856078804233748788580873623628726 0081631789056

Což je číslo s 268 ciframi a označme si jej IV_{dec}. Vhodná hodnota základu r zůstává 3,94 a stejně tak i hodnota základu x je 0,4. Hodnoty r a x v závislosti na IV jsou vypočteny pomocí následujících rovnic.

$$r = 3{,}94 + IV_{dec} * 10^{-270}$$

$$x = 0{,}4 + IV_{dec} * 10^{-269}$$

Tímto dosáhneme, že pro každé jednotlivé IV bude chaotický pseudo-náhodný generátor v rámci režimu činnosti PM-DC-LM „nastartován“ rozdílnými parametry r a x. Tudíž hodnoty r a x budou reálná čísla s přesností na 271 (v případě r) a 270 (v případě x) platných číslic a budou pro každé IV jedinečná. Dokonce v porovnání s předchozí variantou máme mnohem větší prostor pro volbu pozic čísel d a g v rámci generátoru. Celkově i při vyvarování počítání čísel v rámci typu float máme vhodnější podmínky pro stabilitu. Finální podoba PM-DC-LM v rámci implementace kryptografického systému TSS lze vidět na následujícím obrázku.

```
novyKlic = bytearray(aktKlic)
for i in range(0, delkaBloku):
    o = aktBlokOtevreny[i]
    s = aktBlokSifrovany[i]
    k = aktKlic[i]
    strX = str(x)
    d = int(strX[-2])
    g = int(strX[-5] + strX[-4] + strX[-3])
    n = nIN
    if d == 1:
        n += (o + s + d + g) % 256
    elif d == 2:
        n += (s + d + g) % 256
    elif d == 3:
        n += (k + d + g) % 256
    elif d == 4:
        n += (2*o - s + d + g) % 256
    elif d == 5:
        n += (2*s + d + g) % 256
    elif d == 6:
        n += (2*k + d + g) % 256
    elif d == 7:
        n += (3*o + 2*s + d + g) % 256
    elif d == 8:
        n += (3*s + d + g) % 256
    elif d == 9:
        n += (3*k + d + g) % 256
    else:
        n += (4*o - 3*s + d + g) % 256
    novyKlic[i] = (n % 256)
    x = NextFromCPRNG(x, r)
return [bytes(novyKlic), x]
```

Obrázek 37 - finální implementace PM-DC-LM

Každopádně i zde platí a mělo by být experimentálně dokázáno, že výsledná hodnota bajtu následujícího klíče je vypočtena s pravděpodobností 1 ku 256 a využití rovnice je co nejvíce vyvážené v závislosti na hodnotách *o*, *s*, *k*, *n*, *d* a *g*. Rovnoměrné využití rovnic je snadné dokázat, protože je určení rovnice závislé na hodnotě d, které může nabývat hodnoty celého čísla na intervalu $\langle 0; 9 \rangle$ → jedna cifra z aktuálně generované hodnoty *x*. Z toho důvodu bylo vybráno 10 rovnic a zároveň se jako směrodatná cifra z čísla *x* bere druhá cifra zprava, která může být 0. Důležité je poznamenat, že hodnota *x* je generovaná pro každý bajt dalšího klíče samostatně. Pokud bychom chtěli otestovat výstupní pravděpodobnostní distribuci výstupních bajtů klíče, tak by to bylo složitější, jelikož je hodnota závislá na aktuálních stavech nejen hodnot *o*, *s*, *k* a *n*, ale i na hodnotách *d* a *g*. Hodnota d testovací prostor až tak neovlivní, ale hodnota g je celé trojciferné číslo, a tudíž může nabývat hodnot na intervalu $\langle 0; 999 \rangle$. Pro určení pravděpodobné distribuční rozložení byl v rámci kapitoly 7.5.3.5 proveden aproximovaný test za vynechání hodnot *g*, která činí v každé z rovnic stejný přírůstek.

V rámci budoucího výzkumu je v plánu pokračovat a vždy je prostor pro optimalizaci a vylepšení. Například by bylo možné:

- **optimalizovat tvary rovnic** – vyšší míra nelineárnosti (obdobně jako rovnice S-Boxu v rámci šifry AES)
- **odvozovat hodnoty *d* a *g* parametricky** – jejich pozici v rámci hodnoty x (závislost na „nasoleném“ *IV* a vstupním nastavení uživatele)
- **optimalizovat základní hodnoty *r* a *x*** – analýza a vyvarování se místům s nízkou mírou chaotického chování
- **převést podobu tak, aby byla aplikovatelná na stávající blokové šifry**
- **a mnoho dalšího.**

7.5.3 Testování režimů činnosti PM

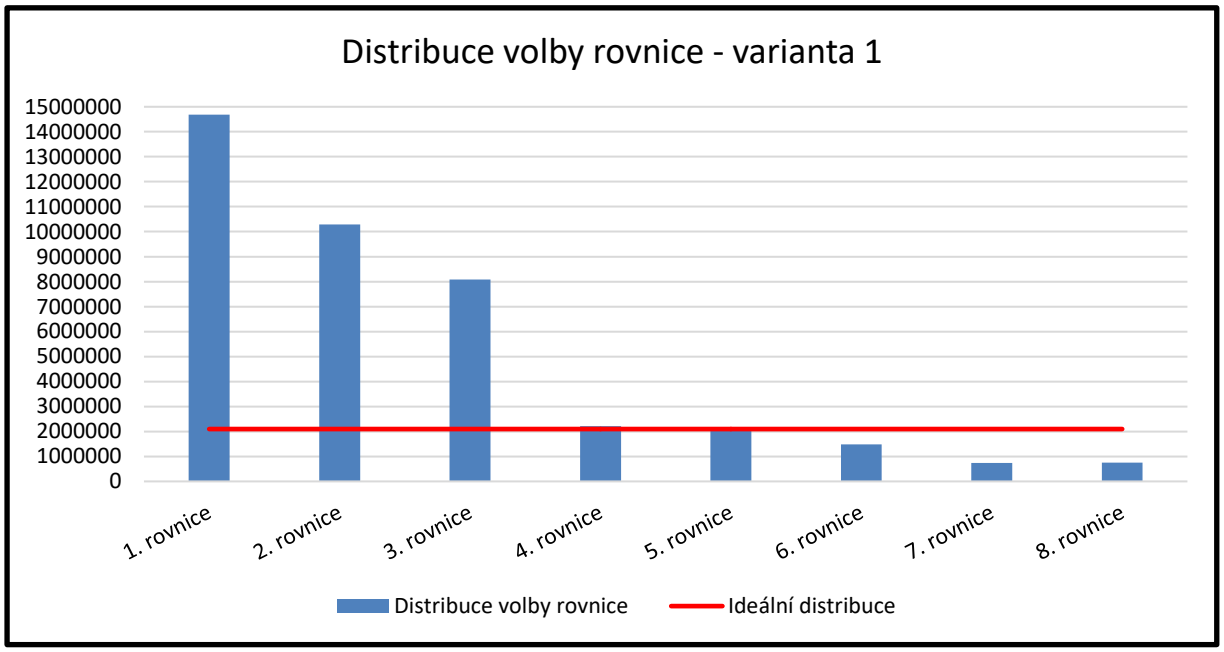
Tato kapitola je věnována testování navržených pěti variant režimů činnosti blokových šifer. Testování bylo zaměřeno hlavně na ověření distribučního rozložení využití rovnic v závislosti na vstupních hodnotách *o*, *s* a *k*. Dále na ověření vhodné distribuce výstupních hodnot bajtů klíče z rovnic vůči vstupním hodnotám – *o*, *s*, *k* a *n* → vhodnost volby tvaru rovnic. Všechny testy proběhly při volbě všech kombinací vstupních hodnot.

7.5.3.1 Testování režimu činnosti PM – varianta 1

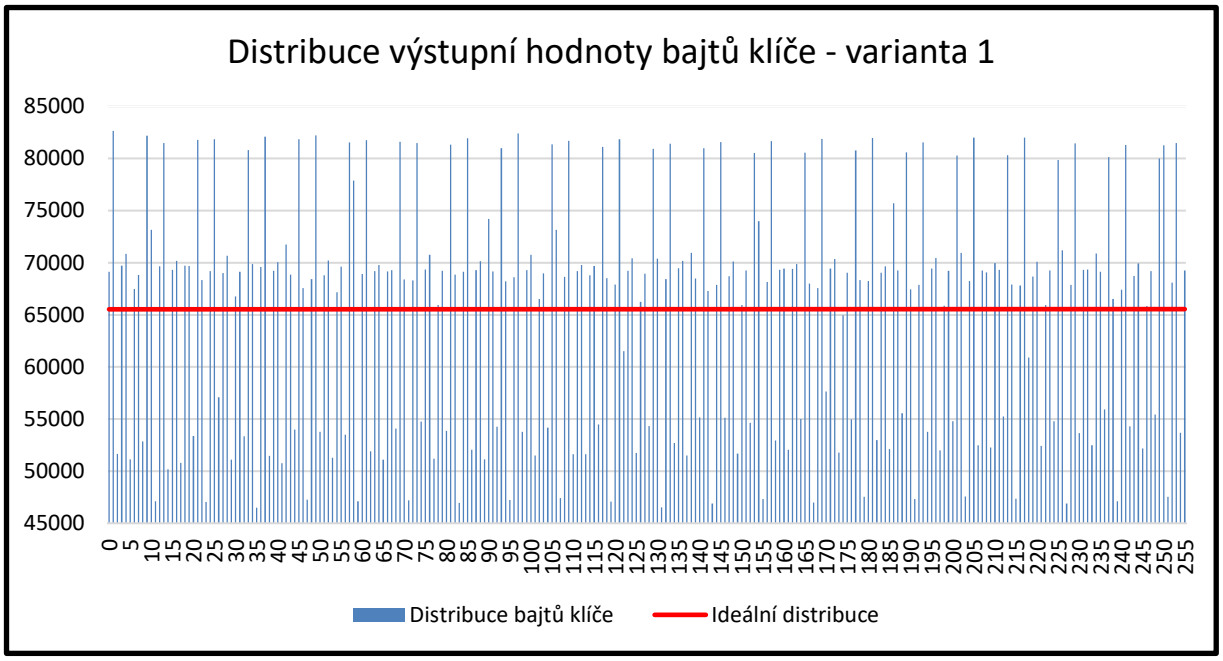
Testování varianty 1, tedy původního návrhu z diplomové práce probíhalo za volby všech kombinací následujících parametrů:

- o - hodnota bajtu předchozích otevřených dat $\rightarrow \langle 0, 255 \rangle$
- s - hodnota bajtu předchozích šifrových dat $\rightarrow \langle 0, 255 \rangle$
- k - hodnota bajtu posledního využitého klíče $\rightarrow \langle 0, 255 \rangle$

Jak testování rozložení zvolené rovnice, tak i testování distribuce výstupního bajtu probíhalo za vyzkoušení všech kombinací vstupů. Jednalo se o vyzkoušení všech kombinací vstupů $\rightarrow 256^3 = 16777216$. Pro ideální rozložení při výběru použité rovnice bylo cílem dosáhnout, aby každá z rovnic byla využitá s pravděpodobností jedna ku osmi $\rightarrow$ každá rovnice by měla být využitá právě 2097152 krát. Volba rovnice byla na základě hodnot *soucin* $\rightarrow$ viz obrázek číslo 28. Při distribuci hodnot výstupní hodnoty klíče bylo cílem dosáhnout volby bajtu s pravděpodobností jedna ku dvě stě padesáti šesti $\rightarrow$ každý bajt by měl být výstupem právě 65536 krát. První graf znázorňuje počty využití rovnic a druhý graf distribuci hodnot bajtů klíče na výstupu.



Obrázek 38 - Distribuce volby rovnice - varianta 1 (vlastní)



Obrázek 39 - Distribuce výstupní hodnoty bajtů klíče - varianta 1 (vlastní)

Na základě výše uvedených grafů a distribucí můžeme vidět, že varianta 1 není ideální a ani se ideálnímu stavu neblíží. Uvedené výsledky byly důvodem tvorby dalších variant.

7.5.3.2 Testování režimu činnosti PM – varianta 2

Testování varianty 2, tedy po optimalizaci s cílem vylepšit rozložení distribuce volby rovnice probíhalo za volby všech kombinací následujících parametrů:

- o - hodnota bajtu předchozích otevřených dat $\rightarrow \langle 0;\ 255 \rangle$
- s - hodnota bajtu předchozích šifrových dat $\rightarrow \langle 0;\ 255 \rangle$
- k - hodnota bajtu posledního využitého klíče $\rightarrow \langle 0;\ 255 \rangle$
- n – hodnota parametru N $\rightarrow \langle 0;\ 255 \rangle$

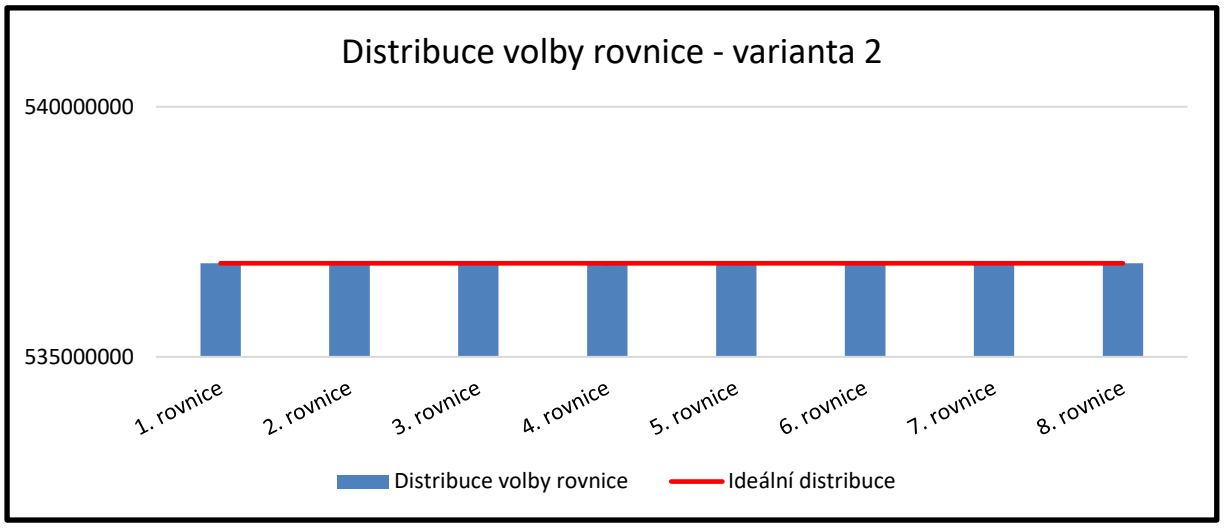
Zde došlo ke změně volby rovnice hlavně v podobě parametru *soucin* na *soucet* (viz obrázek číslo 28) → volba rovnice dle následujícího vzorce.

$$zvolenaRovnice = (o + s + k)\ mod\ 8$$

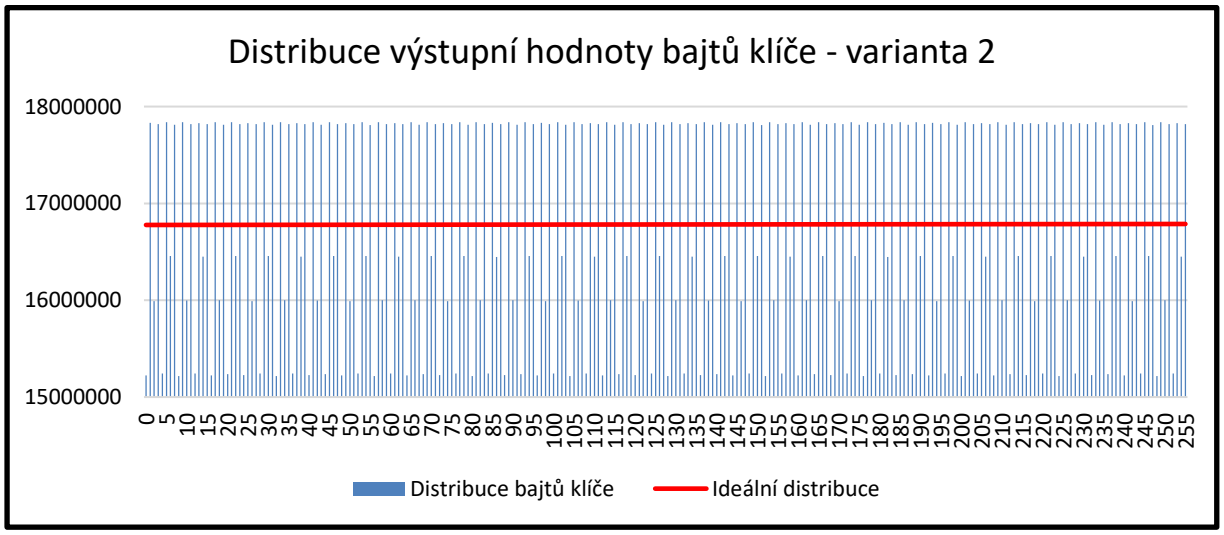
Kdy parametr *zvolenaRovnice* nabývá hodnoty celého čísla na intervalu $\langle 0; 7 \rangle$.

Otestovaná distribuce volby rovnice, jako i distribuce výstupní hodnoty bajtu klíče bylo provedeno za vyzkoušení všech kombinací hodnot o, s, k a n. Počet kombinací vstupů byla $256^4 = 4294967296$. Pro ideální distribuci volby

rovnice je cílová hodnota 536870912 → využití rovnice. Pro ideální distribuci výstupních hodnot bajtů klíče je hodnota 16777216. První graf zobrazuje distribuci volby rovnice a druhý graf distribuci výstupních hodnot bajtů klíče.



Obrázek 40 - Distribuce volby rovnice - varianta 2 (vlastní)

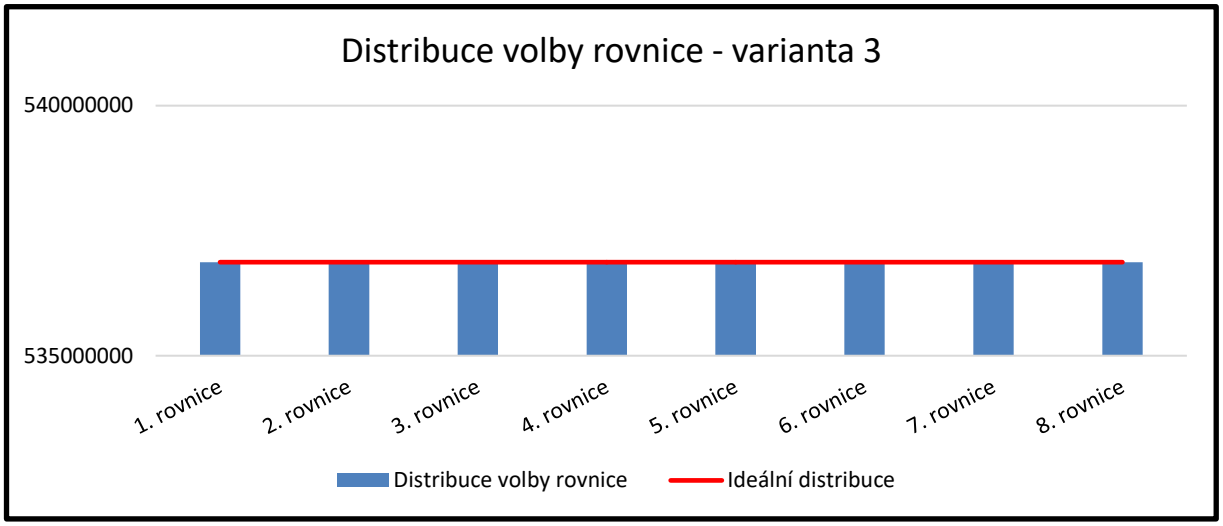


Obrázek 41 - Distribuce výstupní hodnoty bajtu klíče - varianta 2 (vlastní)

Z prvního grafu je možné vidět, že distribuce volby rovnice v rámci režimu ve variantě 2 je ideální → 536870912. Rozložení výstupních hodnot bajtů klíče, jak můžeme vidět z druhého grafu výše, není ideální, ale mírně vylepšené v porovnání s variantou 2. Kdy v porovnání s ideální hodnotou 16777216 se hodnoty pohybují na intervalu ⟨15216645; 17838645⟩.

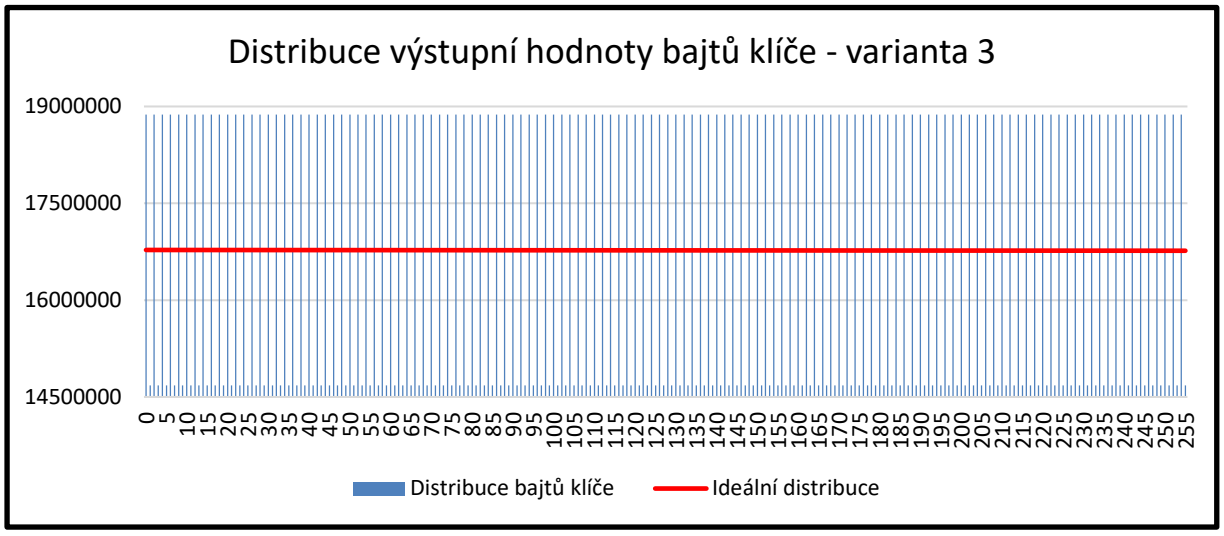
7.5.3.3 Testování režimu činnosti PM – varianta 3

Testování varianty 3 probíhalo stejně jako testování varianty 2. Rozdíl ve variantě 3 a variantě 2 je pouze ve změně tvaru dílčích rovnic.* Pro úplnost, otestovaná distribuce volby rovnice, jako i distribuce výstupní hodnoty bajtu klíče bylo provedeno za vyzkoušení všech kombinací hodnot o, s, k a n. Počet kombinací vstupů byla $256^4 = 4294967296$. Pro ideální distribuci volby rovnice je cílová hodnota 536870912 → využití rovnice. Pro ideální distribuci výstupních hodnot bajtů klíče je hodnota 16777216. První graf zobrazuje distribuci volby rovnice a druhý graf distribuci výstupních hodnot bajtů klíče.



Obrázek 42 - Distribuce volby rovnice - varianta 3 (vlastní)

* *Při vykonávání testů se potvrdil nárůst výpočetního výkonu → odvození bajtu klíče bylo zde průměrně rychlejší, jak při využití varianty 2.*



Obrázek 43 - Distribuce výstupní hodnoty bajtu klíče - varianta 3 (vlastní)

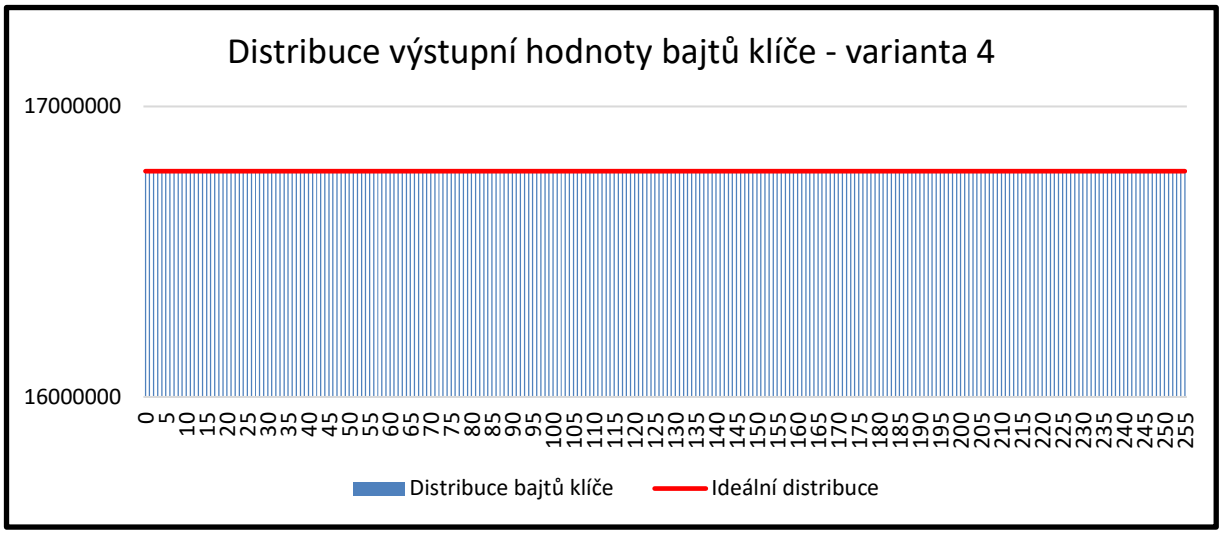
Z grafů výše je patrné následující. První graf je stejný, protože nedošlo ke změně vzorce pro určení rovnice. Druhý graf ukazuje, že tvar rovnic hraje principálně velkou roli v distribučním rozložení. Můžeme pozorovat, že minimum a maximum bylo vyšší, ale oscilace mezi hodnotami je stabilnější. V podstatě lze říci, že sudé hodnoty bajtu (včetně hodnoty 0) byly generované s vyšší pravděpodobností než liché hodnoty bajtů. Každá lichá hodnota bajtu byla generována právě 14680064 krát a hodnota sudého bajtu (včetně hodnoty 0) byla vygenerována právě 18874368 krát.

7.5.3.4 Testování režimu činnosti PM-PolyUltra – varianta 4

Testování varianty 4 → PM-PolyUltra probíhalo se stejnými parametry jako varianty 2 a 3. Nicméně nebylo třeba testovat rozložení volby rovnice, protože varianta 4 využívá pouze jedné rovnice → výstupní bajt klíče je odvozen ze součtu všech parametrů o, s, k a n za modulace hodnotou 256. Viz následující rovnice.

$$hodnotaVýstupníhoBajtu = (o + s + k + n)\ mod\ 256$$

Předpokladem bylo dosažení vyrovnané (ideální) distribuce výstupní hodnoty bajtu klíče. Jak ukazuje následující graf, tak předpoklad byl potvrzen. Ideální distribuce pro každý bajt je 16777216, čehož varianta 4 dosahuje.

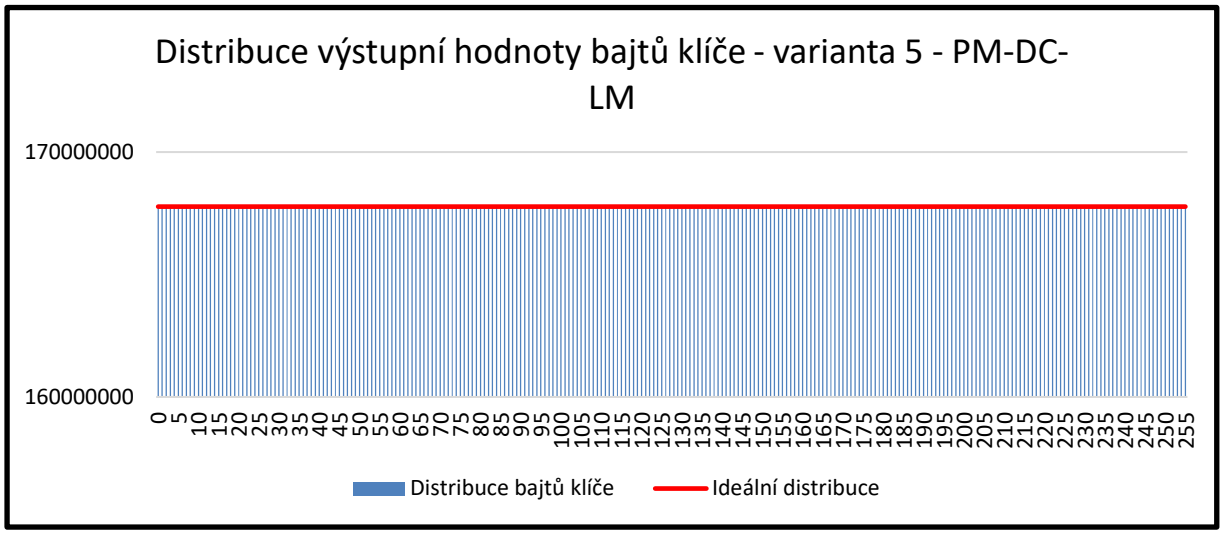


Obrázek 44 - Distribuce výstupní hodnoty bajtu klíče - varianta 4 - PolyUltra (vlastní)

Zároveň se jedná výpočetně nejvýkonnější variantu ze všech testovaných.

7.5.3.5 Testování režimu činnosti PM-DC-LM – varianta 5

Testování varianty 5 → PM-DC-LM probíhalo za mírně odlišných podmínek. Bylo potřeba vzít v potaz větší množinu vstupních hodnot, protože v rámci výpočtu výstupního bajtu klíče je zapojená hodnota *d* (výstup chaotického pseudo-náhodného generátoru čísel), která je zároveň využita jako řídící hodnota volby rovnice pro výpočet hodnoty výstupního bajtu klíče. Proto kromě všech kombinací hodnot *o*, *s*, *k* a *n*, jako v případě testování variant 2 a 3, byla využita i hodnota *d* → hodnota na intervalu $\langle 0; 9 \rangle$. Ve výsledku bylo potřeba otestovat $10 \cdot 256^4 = 42949672960$ kombinací vstupů. Hodnota g byla přitom zanedbána, jak bylo uvedeno výše. To z důvodu, že by nebylo možné otestovat všechny kombinace, kterých by bylo 1000 krát více, ale zároveň proto, že hodnota g vstupuje do každé rovnice stejně a nezávisle na hodnotách *o*, *s*, *k*, *n* a *d* -> tudíž je ve formě konstanty a nemá na výstupní distribuci hodnot bajtů vliv. Výsledná očekávaná ideální distribuce výstupních hodnot bajtu klíče by měla být 167772160 → 10 krát vyšší jako v případě variant 2 a 3. Jelikož byla rovnice zvolena ve vazbě na hodnotu d (nebyla zde rovnice v závislosti na hodnotách *o*, *s* a *k*, jako v případě variant 2 a 3), tak nebylo třeba testovat distribuci volby rovnice. Každá rovnice byla zvolena pro každou hodnotu *d* stejně krát → 4294967296 krát. Následující graf zobrazuje výslednou distribuci výstupních hodnot bajtu klíče.



Obrázek 45 - Distribuce výstupního hodnoty bajtu klíče - varianta 5 - PM-DC-LM (vlastní)

Jak lze vidět z grafu, tak varianta 5 → PM-DC-LM dosahuje jak ideální distribuce volby rovnice, tak ideální distribuce výstupních hodnot bajtu klíče.

7.5.4 Shrnutí

Pro testy je potřeba říct, že byly zvolené optimální podmínky → ideální distribuce vstupních hodnot, kdy každá z hodnot byla použita právě jednou; se stejnou pravděpodobností. V praxi by byla skutečnost mírně odlišná a bude ovlivněna následujícími faktory:

- Entropií otevřených dat
- Entropií šifrových dat
- Entropií klíčových dat – s tím související
  - Entropií odvozeného *IV*
  - Entropií hodnot *d* a *g* v rámci režimu PM-DC-LM
- Uživatelským nastavením

Entropie chaotického pseudo-náhodného generátoru čísel v rámci PM-DC-LM a s tím související entropie vygenerovaných hodnot *d* a *g* byla z větší části zajištěna vhodným nastavením počátečních hodnot *r* a *x*. Zároveň byla entropie zvýšena tím, že byl vytvořen vhodný proces převodu *IV* na hodnoty *r* a *x*. Při aplikaci generátoru v praktické implementaci, před generováním prvních hodnot *d* a *g*, by bylo vhodné jej nechat v několika iteracích běžet „naprázdno“. Tím bude docílena vyšší míra chaotičnosti systému.

Entropie otevřených dat, šifrových dat a klíčových dat byla zvýšena pomocí solení. Při využití soli pro „předsolení“ byla zvýšena entropie otevřených dat, kdy

současně došlo ke zvýšení míry difúze systému. Šifrová data a klíčová data jsou ovlivněna zejména při „nasolení“ odvozeného *IV* z uživatelských klíčových dat. Lze tedy tvrdit, že ve výsledku bude entropie systému velmi závislá na entropii generovaných solí. V neposlední řadě zvýšení entropie šifrových dat obstará bloková šifra a šifrovací proces.

Všechny varianty výše jsou ve své finální verzi prototypu, nikoliv nejlepší praktickou aplikací. Na základě výsledků z předchozí kapitoly můžeme tvrdit, že pro praxi se nejvíce jako vhodné jeví varianta 4 → PM-PolyUltra a varianta 5 → PM-DC-LM. Varianta 4 je vhodnější při potřebě výkonu, kdy varianta 5 je vhodnější při požadavku komplexnějšího (variabilnějšího) chování.

Celkově jsou tyto poznatky vhodným základem pro další výzkum → optimalizace rovnice pro volbu rovnice za ideální distribuce výstupních hodnot bajtů klíče. Výzkum ukázal, že tyto dva faktory hrají roli.

8 TESTOVÁNÍ KRYPTOGRAFICKÉHO SYSTÉMU TSS

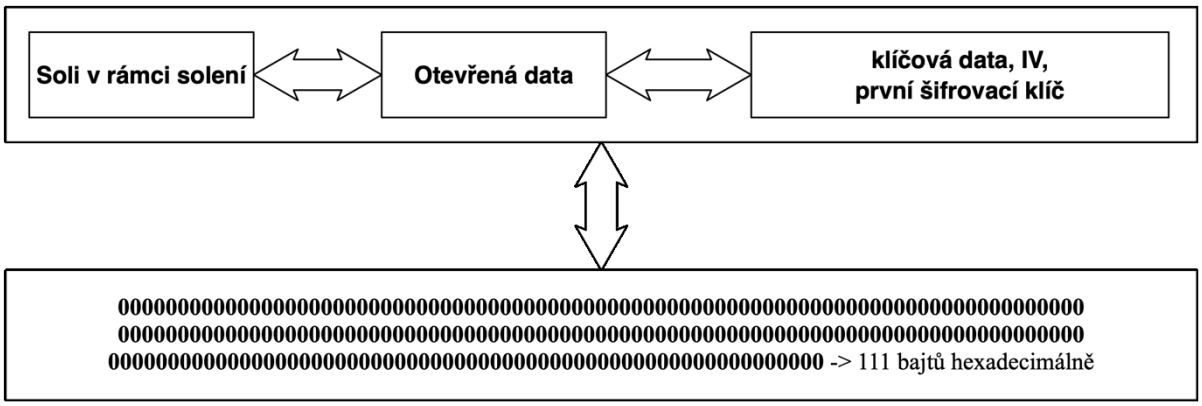
Následující kapitola se věnuje testování vytvořeného systému TSS jako souhrnu všech součásti systému jako celku. Kapitolu jde rozdělit do testování systému TSS bez využití solení (bez generování solí a za využití solí pouze s nulovými bajty) a na testování kompletního systému TSS včetně solení. Další podkapitoly se věnují testování rychlosti šifrování v porovnání se šifrou AES a jako poslední podkapitola je sumarizace výsledků testování systému TSS.

Konkrétní podmínky testování a obsah testů jsou uvedeny v rámci jednotlivých podkapitol. Všechny entropie jsou vypočteny z hexadecimálního tvaru dat, a tudíž entropie v rámci následujících podkapitol a provedených testů může nabývat hodnotu reálného čísla na intervalu $\langle 0; 4 \rangle$.

8.1 Testování TSS bez využití solení a využití klíčových dat

První část testování je věnovaná testům bez využití solení. To znamená, že testy byly provedeny v rámci systému TSS, ale soli nebyly náhodně vygenerované, ale upraveny na hodnoty samých nul. Znamená to, že sůl pro „nasolení“ *IV*, pro „předsolení“ i pro „zasolení“ otevřených dat byly nulové a vždy měli délku 111 bajtů. Podobně parametry pro solení byly nastaveny ve všech testech na hodnoty nula.

V rámci následujících testů byla klíčová data a následně klíč k šifrování prvního bloku dat nastavené na hodnotu nulových bajtů o délce 111 bajtů. Stejně byla nastavená otevřená data jako nulová o délce 111 bajtů a soli pro solení (viz digram níže).



V rámci prvního testování byly uživatelské vstupní parametry pro variabilitu systému TSS nastaveny nejprve jako nulové; konkrétně následovně (podrobný popis parametrů viz kapitola 7.4):

- Parametry pro výpočet počet rund
  - Defaultní počet rund → 0
  - Defaultní řídící proměnná počtu rund → 0
- Parametry pro výpočet proměnné N
  - Defaultní hodnota proměnné N → 0
  - Defaultní řídící proměnná proměnné N → 0
- Parametry pro výpočet délky bloku
  - Defaultní řídící proměnná pro výpočet délky bloku → 0
  - Defaultní hodnota variability délky bloku → 0
  - Defaultní řídící proměnná pro výpočet variability délky bloku → 0
- Parametry pro odvození nastavení proměnné X
  - Defaultní hodnota proměnné X → 0
  - Defaultní řídící proměnná proměnné X → 0
- Parametry přesnosti pro režim činnosti PM-DC-LM (viz kapitola 7.4.7)
  - Defaultní přesnost → 0
  - Defaultní řídící proměnná pro přesnost → 0

Dále byly parametry přenastaveny na hodnoty jedenáct; konkrétně:

- Parametry pro výpočet počet rund
  - Defaultní počet rund → 11
  - Defaultní řídící proměnná počtu rund → 11
- Parametry pro výpočet proměnné N
  - Defaultní hodnota proměnné N → 11
  - Defaultní řídící proměnná proměnné N → 11
- Parametry pro výpočet délky bloku
  - Defaultní řídící proměnná pro výpočet délky bloku → 11
  - Defaultní hodnota variability délky bloku → 11
  - Defaultní řídící proměnná pro výpočet variability délky bloku → 11
- Parametry pro odvození nastavení proměnné X
  - Defaultní hodnota proměnné X → 11
  - Defaultní řídící proměnná proměnné X → 11
- Parametry přesnosti pro režim činnosti PM-DC-LM
  - Defaultní přesnost → 11
  - Defaultní řídící proměnná pro přesnost → 11

Testy byly provedeny pro ukázku souvislosti mezi výslednými šifrovými daty a mezi nastavením vstupních parametrů. Bylo provedeno celkem 6 testů → 3 pro nulové a 3 pro parametry s hodnotou 11. Pro každý režim činnosti zvlášť, kdy byly zvoleny varianty 3, 4 a 5 (viz kapitola 7.5.3). Pro potřeby prezentace

jsou zde rozepsané a porovnané výsledky testování s nastavením nulových parametrů proti nastavení parametrů na hodnotu 11, kdy byl v obou případech požit režim činnosti PM-DC-LM → varianta 5. Souhrnnou sumarizaci v rámci tabulky číslo XXX.

Při nastavení nulových parametrů a volbě režimu činnosti PM-DC-LM byly výsledná šifrová data v hexadecimálním tvaru následovná (pro potřeby textu označme jako šifrová data 1):

0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000006a699626b0f16b1fc05f6fb0f8d4ab97eb71b
35b1d731ac9915d165423c20e103aaaa741cd45c3b73547e94b54f2e503d880e43
f2911a627191a8ec5e210f10e3ec0db0b06d606757ae114c4a559488b5b67ecae0
29a7b6cf49b0c6098b78bbac97602191eb8d2f3170ebaa388269a81ed07a9953d3
139dc850d40082382162323d65f66c578544c27866362ab525bc989eb601754ba
0fa5e69852b2f8752b475dd0033e81b82602c7dafef320a02bf90d28cc39faa28823
6910582d741b43f286b62fc396ff385571e31f31e730630f5b129ec6b17af1c71e5c
cb71274235466fbf36ce5e52e134df45605bf6bd551ddf4edf3aca1f93d92fca078fb
b19ba6a230db0ef613de77916f0df5bcc343f246d7afeed33d6ce72332b49cf93c6e
09f4964581206e06283d94b8a47c8c3edfe05440c5a88b8afb8983e52740c0e9fd7
0b65e20546379c097e5e46b338$_{hex}$

Při přenastavení parametrů na hodnoty 11 a volbě režimu činnosti PM-DC-LM byly výsledná šifrová data v hexadecimálním tvaru následovná (pro potřeby textu označme jako šifrová data 2):

0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000006a0ba7565dd32d8fbb581f046de2577a65a
dedf25abead97b0272f391f9c535149c25c916d985176c1a450fe3518cf16bd8075e
17cc1bd6d56a3e1ac7116101f46506a056772360d93b7a5f3816c7ec7c848e3348
61c269a1276d199fee09b350616637536e09b72096180466568882 07c755afa599
3a5902be0cfbcd3200dfba954b2516d29d9fa20a55786eaa59a6ec591b0ae672e15
49fd00c24c8ba3d682ece37627e78d6850159fdb87711aca01120f44ade0b6ba56a
1e3f415e51d34d31676b479c136eca53638fa4c0a6dcb5343daf231c56aaa1d62d0
0123dc2f8dcb3e83988f2054035b14560c1bdaef66cd8ee397d373fdf7ca1ba8bf90
0f3f24e800ecd95ebdfaeb2896ccefe9340107f29ccf924fa256eb30a6c3964418ce6
819aae5b0e1971831d5f2459860724bdc718e3fb4ae9615f951525a9d15dd6ea01
5ee23e5e1e401c850e3b8511200$_{hex}$

V rámci šifrových dat 1 a 2 barevné znázornění vyjadřuje:

- Žlutá barva → zašifrovaná sůl pro „nasolení“
- Zelená barva → zašifrované soli pro „předsolení“ a „zasolení“
- Červená barva → zašifrovaná otevřená data*

Pozice soli pro nasolení vzhledem nastavení procesu pro účely testů bude vždy na pozici nula → na začátku šifrových dat. Souvisí to výpočtem pozice soli z *IV* a jelikož bylo zvoleno nulové, tak i pozice soli pro „nasolení“ bude vždy nulová. V rámci testování v kapitole 8.2 je vidět, že při plném nastavení systému TSS je pozice soli variabilní a nemusí být nulová.

Ze šifrových dat 1 a 2 lze vidět, že při změně parametrů pro variabilitu systému TSS z hodnot 0 na hodnoty 11 došlo k zásadnímu ovlivnění výsledných šifrových dat. Konkrétně se jedná o změnu 320 bajtů šifrových dat 2 v porovnání se šifrovými daty 1. Procentuálně vyjádřeno bylo změněno v rámci šifrových dat 2 přibližně 96,97 % bajtů v souvislosti se změnou parametrů.

Pokud bychom se podívali na šifrová data, kdy byly parametry nastaveny na hodnoty 11, ale pro šifrování byl využitý režim činnost PolyUltra, tak vypadají následovně (pro potřeby textu označme jako šifrová data 3):

000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000006a0ba7565dd32d8fbb581f046de2577a65ad
edf25abead97b0272f391f9c535149c25c916d985176c1a450fe3518cf16bd8075e
17cc1bd6d56a3e1ac7116101f46506a056772360d93b7a5f3816c7ec7c848e3348
61c269a1276d199fee09b350616637536e09b720961804665688294301ee0fc5f
9db79e86ddc02ff7236656a13aa9859baec4522fed36d3f6c696f0a4ec836d39859
be3015b63e314b22310bca173835c72a4c320ee303e3c4ecdb6c8eb3c7e98a491d
5cfe8cb063b5623556e1b72c85220e73251468e7c0b0efa5dbd698446f57f42ce85
8ddd004da370aad74656d95f9a60f1a11b59ce7336e602c3ceb497b969050edfd2b
c477f483d366212f2ced0511e22d0e3e87be8892d80048db6a07f26397987d6a52

---

* *Barevná reprezentace neznamená rozdělení na šifrované bloky, protože počet bloků souvisí s parametrizací systému (viz kapitola 7.4 a logy v rámci přílohy PPP)*

3b226c301d8806f21abb34a6b1fa25e7dc45af7cc95e9c71d6d894313de6f24d5f1953feaaaf05c9d6b2974285933df59d$_{hex}$

Při porovnání šifrových dat 3 proti šifrovaným datům 2 lze vidět, že zašifrovaná sůl pro „předsolení“ je v obou případech zašifrována stejně → jejich šifrová data jsou totožná. Je to způsobeno tím, že parametry byly v obou případech nastaveny na hodnoty 11 a změna režimu činnosti se projeví až v případě šifrování druhého bloku otevřených dat (zde otevřená data). Konkrétně v rámci systému TSS je pro šifrování dalšího bloku klíč odvozen odlišně → ve vazbě právě na zvolený režim činnosti. Pokud bychom procentuálně vyjádřili počet změněných bajtů šifrových dat 3 oproti šifrovým datům 2 (pouze z části za solí pro „předsolení“ → poslední 222 bajtů), tak bude změna 94,14 %. To znamená že bylo změněno 209 bajtů z 222 šifrovaných.

Kvalitu systému TSS i bez využití solení lze vidět na následujících vypočtených entropiích.[8] Vypočtené entropie jsou pro šifrová data z testů, kdy byla vypočtená entropie pro kompletní šifrový text, ale i pro část bez prvních 111 bajtů → soli pro „nasolení“. Sůl pro „nasolení“ lze pro výpočet entropie zanedbat hlavně z toho důvodu, že podmínky testů byly nastaveny, že se nejedná o náhodná data a že sůl nevstupovala do procesu šifrování. Jelikož jsou data nulová, tak je pochopitelné, že budou mít negativní vliv na vypočtené entropie. Entropie, jak bylo uvedeno v úvodu kapitoly, byly vypočteny z hexadecimálních dat, kdy lze očekávat hodnoty reálných čísel na intervalu $\langle 0; 4\rangle$, kdy hodnota 4 značí nejvyšší entropii a cílem systému je, aby se entropie šifrových dat k této maximální hodnotě blížila → vyjádření hodnoty entropie je na 11 platných číslic.

[8] *Pro větší směrodatnost výpočtu entropie by byla vhodnější delší šifrová data. Podrobnější testy jsou provedeny v kapitole 8.2 včetně solení*

Tabulka 3 - Výpočet entropie TSS s nulovými solemi a se změnou parametrů

Entropie -> TSS s nulovými solemi a změnou parametrů		
Varianta	Entropie - celá data	Entropie - bez "nasolení"
TSS + PM - parametry 0	3,6005585670	3,9890066218
TSS + PM-PolyUltra - parametry 0	3,6104553882	3,9889534982
TSS + PM-DC-LM - parametry 0	3,5979363475	3,9892705569
TSS + PM - parametry 11	3,6013654878	3,9824469681
TSS + PM-PolyUltra - parametry 11	3,6104553882	3,9788065314
TSS + PM-DC-LM - parametry 11	3,5893556851	3,9740693394

8.2 Testování entropie systému TSS včetně solení

V rámci této kapitoly jsou rozepsané výsledky z testování entropie celého systému TSS včetně aplikace solení. Kapitola dále porovnává entropie nejen entropie výsledného systému TSS, včetně všech výsledných variant režimů činnost, ale i se šifrováním pomocí šifry AES v různých kombinacích režimů činnosti. V neposlední řadě jsou zde uvedeny i entropie generovaných dat z generátorů čísel. V rámci TSS se jedná o kombinace:

- TSS + PM (varianta 3, viz kapitola 7.5.3.3)
- TSS + PM-PolyUltra (varianta 4, viz kapitola 7.5.3.4)
- TSS + PM-DC-LM (varianta 5, viz kapitola 7.5.3.5)

V případě šifry AES s délkou klíče 256 bitů o varianty:

- Klasické režimy činnosti
  - AES256-ECB
  - AES256-CBC
  - AES256-CTR
  - AES256-CFB
  - AES256-OFB
- Moderní režimy činnosti (AEAD)
  - AES256-CCM
  - AES256-GCM
  - AES256-EAX
  - AES256-OCB

V případě generování náhodných dat byly použité dva generátory čísel na srovnání, a to konkrétně:

- Funkce *urandom()* jako základní vestavěná funkce jazyka Python 3:7.6 knihovny os

- Funkce *get_random_bytes()* v rámci kryptografické knihovny PyCryptodome v poslední verzi

Testování probíhalo v rámci programovacího jazyka Python verze 3.7.6 na stejném HW a za stejným podmínek. Vždy byla šifrována otevřená data o délce 1 111 111 bajtů a šifrování bylo provedeno ve 111 iteracích. Testování bylo provedeno ve dvou bězích. V prvním běhu byla šifrována otevřená data, kdy všechny bajty měly hodnotu 0 a klíče měly také pouze nulové bajty → v případě TSS 111 nulových bajtů a v případě AES 32 nulových bajtů. V rámci druhého běhu byly v každé iteraci generovaná náhodná data o délce 1 111 111 bajtů a použitá jako otevřená data. Stejně tak byl pro každou iteraci generovaný nový náhodný klíč → opět pro TSS o délce 111 bajtů a pro AES o délce 32 bajtů. *IV* v rámci šifry AES a soli v rámci TSS nebyly nijak ovlivněny. Pro režimy moderní režimy činnosti CCM, GCM, EAX a OCB bylo potřeba nastavit hodnotu „headeru“ (viz dokumentace knihovny PyCryptodome). V prvním běhu byla zvolena jako 111 nulových bajtů a pro potřeby druhého běhu byl vždy pro každou iteraci generován nový náhodný header.[9]

Výsledky jsou vidět v následujících dvou tabulkách. Všechny entropie byly vypočteny z hexadecimální reprezentace šifrových dat a jsou zobrazeny s přesností na 11 platných číslic. V souvislosti s výpočtem entropie z hexadecimálního tvaru dat lze očekávat, že by se hodnoty entropií měly blížit k maximální hodnotě 4.

[9] *Pro šifru AES byly hodnoty generované pomocí funkce get_random_bytes() a pro TSS pomocí funkce urandom()*

Tabulka 4 - Testování entropie TSS vs. AES - nulové hodnoty OT a klíče (vlastní)

Entropie -> nulové hodnoty OT a klíče					
Varianta	Počet iterací	Délka OT (B)	Entropie - průměr	Entropie - min	Entropie - max
TSS + PM	111	1111111	3,9999950574	3,9999897006	3,9999982711
TSS + PM-PolyUltra	111	1111111	3,9999948037	3,9999889434	3,9999983851
TSS + PM-DC-LM	111	1111111	3,9999952784	3,9999873040	3,9999980147
AES256-ECB	111	1111111	3,2807236356	3,2807236356	3,2807236356
AES256-CBC	111	1111111	**3,9999952832**	3,9999896777	3,9999986514
AES256-CTR	111	1111111	3,9999948563	3,9999899455	3,9999982716
AES256-CFB	111	1111111	3,9999952179	3,9999899586	**3,9999989392**
AES256-OFB	111	1111111	3,9999951129	3,9999892935	3,9999983596
AES256-CCM	111	1111111	3,9999948921	3,9999898791	3,9999985659
AES256-GCM	111	1111111	3,9999952123	3,9999895944	3,9999985018
AES256-EAX	111	1111111	3,9999949681	3,9999899133	3,9999981307
AES256-OCB	111	1111111	3,9999947250	3,9999898085	3,9999980943
Maximální průměrná entropie -> **AES256-CBC**			**3,9999952832**		
Maximální entropie -> **AES256-CFB**					**3,9999989392**

Jak je možné vidět z první tabulky, tak s nejlepší průměrnou entropii při šifrování nulových hodnot bylo šifrování pomocí kombinace **AES256-CBC**. Maximální entropie bylo dosaženo při šifrování kombinací **AES256-CFB**.

Tabulka 5 - Testování entropie TSS vs. AES vs. náhodné generátory - náhodné hodnoty OT a klíčů (vlastní)

Entropie -> náhodné hodnoty OT a klíče					
Varianta	Počet iterací	Délka OT (B)	Entropie - průměr	Entropie - min	Entropie - max
TSS + PM	111	1111111	3,9999950738	3,9999908275	3,9999984169
TSS + PM-PolyUltra	111	1111111	3,9999949853	3,9999896968	3,9999979634
TSS + PM-DC-LM	111	1111111	3,9999952609	3,9999903583	**3,9999989922**
AES256-ECB	111	1111111	3,9999948387	3,9999903530	3,9999982111
AES256-CBC	111	1111111	3,9999952414	3,9999891625	3,9999986157
AES256-CTR	111	1111111	3,9999950330	3,9999885891	3,9999978544
AES256-CFB	111	1111111	3,9999951390	3,9999890055	3,9999983995
AES256-OFB	111	1111111	3,9999949671	3,9999904922	3,9999982140
AES256-CCM	111	1111111	3,9999951344	3,9999872249	3,9999983546
AES256-GCM	111	1111111	3,9999951228	3,9999907588	3,9999987934
AES256-EAX	111	1111111	3,9999953396	3,9999888946	3,9999989337
AES256-OCB	111	1111111	3,9999950577	3,9999897997	3,9999981906
Random PyCryptodome	111	1111111	3,9999951461	3,9999896907	3,9999981823
Random urandom()	111	1111111	**3,9999953764**	3,9999910962	3,9999987336
Maximální průměrná entropie -> **urandom()**			**3,9999953764**		
Maximální entropie -> **TSS + PM-DC-LM**					**3,9999989922**

V případě šifrování náhodných dat, náhodným klíčem vyšla „lépe“ kombinace **TSS + PM-DC-LM** s dosaženou maximální entropií 3,9999989922.

Nejlepší průměrná entropie byla po 111 iteracích dosažena při generování náhodných dat pomocí funkce *urandom()*.

Na základě entropií jde usuzovat kvalitní míru difúze a konfúze systému TSS, zejména při šifrování nulových bajtů otevřených dat. Dále lze říci, že se šifrová data blíží datům náhodným, kdy náhodnost dat by byla ještě nutná podrobněji otestovat, například za využití Diehard testů. Celkově z testování lze říci, že jak systém TSS a AES jsou na srovnatelné úrovni. Rozdíly v dosažených entropiích jsou nepatrné a nelze tvrdit, že by byl jeden z testovaných lepší či horší. Při dalším spuštění testů by mohl být výsledek jiný. Každopádně TSS i AES dosahují výsledných entropií srovnatelných s entropií náhodně generovaných dat, viz entropie náhodně generovaných dat funkce *urandom()* a *get_random_bytes()*[10] a hodnoty blížící se 4. Jediná kombinace AES256-ECB nedosahovala kvalitní entropie šifrovaných dat při šifrování nulových bloků, což je ale z principu fungování ECB pochopitelné. Výsledná vypočtená entropie znamená vyjádření entropie jednoho zašifrovaného bloku otevřených dat. Zbývající šifrované bloky vypadaly stejně.

8.3 Rychlost šifrování TSS

I za předpokladu, že hlavním cílem nebylo navrhnout kryptografický systém, který by byl rychlý, ale co možná nejbezpečnější a nejkomplexnější, tak bylo provedeno testování rychlosti TSS. Pro představu a porovnání byla zvolena šifra AES s klíčem délky 256 bitů využitím různých režimů činnosti, kterou lze považovat za nejrychlejší šifrovací algoritmus při zajištění dostatečné bezpečnosti. Testy byly provedeny na stejném zařízení:

- MacBook Pro 13 palců, 2017
- Dvoujádrový Intel Core i5 procesor 2,3 GHz
- Operační pamět 16 GB 2133 MHz LPDDR3
- Operační systém macOS Big Sur verze 11.3.1
- Programovací jazyk implementace Python verze 3.7.6

Implementace TSS byla provedena s cílem o co možná největší mírou optimalizace, ale jedná se stále o vzorovou implementaci („nedokonalou“). To

[10] *Obě dvě funkce lze považovat za kryptograficky bezpečné funkce pro generování náhodných čísel [41]; [51]*

v porovnání s implementaci šifry AES a režimu činnosti v rámci knihovny PyCryptodome, která je mnohem lépe optimalizována a lze počítat i s využitím hardwarové akcelerace. Proto už na základě implementace je jasné, že AES má lepší předpoklady pro rychlejší šifrování.

Při šifrování pomocí systému TSS je potřeba si uvědomit, že průběh šifrování není fixní, ale vysoce polymorfní i beze změny vstupních podmínek a parametrů → v závislosti na náhodných solích. Proto doba šifrování nesouvisí pouze s délkou otevřených dat, délkou klíče a komplexností systému, ale doba šifrování se mění ve vazbě na náhodně zvolené parametry jako jsou počet rund, délky šifrovaných bloků, klíč a další.

Pokud budeme šifrovat pomocí šifry AES a jakéhokoliv režimu činnosti, tak doba šifrování bude nezávislá na otevřených datech nebo klíči, ale bude souviset pouze s kvalitou implementace a nastavením → klíč 128, 192 nebo 256 bitů. V kontrastu s tím se doba šifrování pomocí systému TSS bude pohybovat nezávisle na vstupech a implementaci od své minimální doby šifrování po maximální. Pokud bude délka otevřených dat růst, tak i rozptyl doby šifrování mezi minimální a maximální bude vyšší. Doba šifrování TSS tedy souvisí s vygenerovanými náhodných daty ve formě solí.

Pro potřeby testování byl využitý systém TSS v plné podobě (včetně solení) a byly otestovány režimy činnosti PM (varianta 3), PM-PolyUltra (varianta 4) a PM-DC-LM (varianta 5), viz kapitola 7.5. Otevřená data byly nulové bajty o délce 111111 bajtů. Klíčová data byla stejná → 111 nulových bajtů. Pro stanovení průměrné doby testování bylo šifrování provedeno 1111 krát pro každou variantu šifry a režimu činnosti. V případě systému TSS byly parametry zvolené jako nulové.

Následující tabulka shrnuje doby šifrování systému TSS s různými režimy činnosti, kdy lze vidět, že systém TSS v porovnání se šifrou AES je znatelně pomalejší. Ze tří otestovaných variant režimu činnosti byla varianta 4 → PM-PolyUltra průměrně nejrychlejší. V případě šifry AES v kombinaci s režimy činnosti nejsou jsou většinou nepatrné rozdíly až na režim činnosti ECB, který je lehce paralelizovatelný a byl tudíž průměrně nejrychlejší. Naopak režim činnosti CFB byl v průměru nejpomalejší.

Tabulka 6 - Porovnání doby šifrování TSS vs. AES

Doba šifrování							
Varianta	Počet iterací	Délka OT (B)	čas - celkem (s)	čas - průměr (ms)	čas - min (s)	čas - max (s)	kB/s - průměr
TSS + PM	1111	111111	2978,16	2680,61	1,91	7,58	40
TSS + PM-PolyUltra	1111	111111	2927,58	2635,09	1,89	6,69	41
TSS + PM-DC-LM	1111	111111	3755,47	3380,26	2,43	12,08	32
AES256-ECB	1111	111111	0,07	0,06	0,05	0,77	1808447
AES256-CBC	1111	111111	0,32	0,28	0,26	1,57	387524
AES256-CTR	1111	111111	0,22	0,20	0,17	2,82	542534
AES256-CFB	1111	111111	3,73	3,36	3,13	27,83	32294
AES256-OFB	1111	111111	0,34	0,30	0,27	3,49	361689
AES256-CCM	1111	111111	0,60	0,54	0,47	7,47	200939
AES256-GCM	1111	111111	0,35	0,31	0,27	1,72	350022
AES256-EAX	1111	111111	0,80	0,72	0,61	18,74	150704
AES256-OCB	1111	111111	0,34	0,31	0,26	2,02	350022

Na základě výše uvedené tabulky lze odvodit následující:

- Šifrování pomocí TSS ve variantě 3 PM bylo přibližně v průměru 45211 krát pomalejší než AES256-ECB a 807 krát pomalejší než AES256-CFB.
- Šifrování pomocí TSS ve variantě 4 PM-PolyUltra bylo přibližně v průměru 44108 krát pomalejší než AES256-ECB a 788 krát pomalejší než AES256-CFB.
- Šifrování pomocí TSS ve variantě 3 PM bylo přibližně v průměru 56514 krát pomalejší než AES256-ECB a 1009 krát pomalejší než AES256-CFB.

Nejvíce dobu šifrovaní systémem TSS ovlivňuje kombinace počtu rund a délky šifrovaných bloků. Kdy počet rund může být 1 až 4 a délka bloků se může pohybovat v rozmezí 91 bajtů až 111 bajtů. Průměrně tedy lze říci, že v rámci šifrování otevřených dat délky 111 111 bajtů bude šifrováno 1101 bloků, ale v nejhorším případě to může být 1221 bloků, a naopak v nejlepším případě 1001 bloků. Pokud bychom počítali průměrný počet rund 2.5, tak se může jednat o šifrování pomocí 2503 rundami až po šifrování 3053 rundami → nárůst doby šifrování přibližně o 20 procent. Při delších otevřených datech se může jednat o větší rozmezí → viz výpočet počtu šifrovaných bloků kapitola 6.3.4.1.

8.4 Shrnutí

Systém TSS, jak bylo ukázáno výše poskytuje velmi kvalitní pravděpodobnostní distribuci hodnot bajtů šifrových dat → difúzi a konfúzi. Lze vidět, že systém TSS je schopen znemožnit určení jakékoliv vazby mezi šifrovými daty a otevřenými daty či šifrovacím klíčem.

Bylo ukázáno, že i když jsou systémem TSS šifrována otevřená data s velmi nízkou entropií, tak výstupní šifrová data se svou entropií blíží té maximální. Bylo demonstrováno, že systém TSS se chová náhodně polymorfně. To znamená,

parametry systému a průběh šifrování probíhají rozdílně i bez nutnosti měnit vstupní šifrovací klíč (*IV*), uživatelské parametry či otevřená data.

Lze tvrdit, že systém TSS je schopen odolat základním známým kryptoanalytickým metodám, které jsou výhradně aplikovatelné na šifrovací algoritmy a systémy s fixní strukturou. Lze očekávat, že systém je díky jeho náhodně polymorfnímu chování schopen odolat i pokročilým kryptoanalytickým metodám založených na postranních kanálech, útocích založených na časování a v neposlední řadě i aplikaci kvantových počítačů. Teoreticky lze systém TSS brát jako post kvantový symetrický šifrovací algoritmus. To vše za předpokladu další hlubší analýzy → což je možné, při rozsáhlosti faktorů ovlivňující chování systému a jeho komplexnosti, vidět jako zároveň výhodu i nevýhodu.

V neposlední řadě lze vidět výhodu TSS v možnosti flexibilně měnit systém a manipulovat s jeho parametry → délka klíče je pouze jeden z parametrů, který lze jednoduše kdykoliv změnit a s tím související fungování a nastavení celého systému. Jako výhodu a současně nevýhodu je možné považovat výpočetní náročnost (doba šifrování) a nemožnost škálovatelnosti. Výhodu v obtížné aplikaci HW prostředků pro prolomení a nevýhodu v době šifrování.

9 PŘÍNOS PRO VĚDU A PRAXI

Problematikou polymorfních struktur v symetrické kryptografii se věnuje omezené množství publikací. V rámci existujících řešení je pouze malé množství algoritmů, které by byly navrženy v souladu s polymorfním chováním, a i ty jsou polymorfní jen z části – generování substituční tabulky, využití *nonce*/*IV*/tweaku, využití proměnlivé délky bloků, využití variabilních operací (permutace, posun, rotace). Aktuálně využívané šifry, mezi které patří AES, Camellia, TDES/TDEA, Blowfish, Serpent, Twofish, ARIA, IDEA, GOST, RC6 a další mají z většiny fixní průběh – nezávislý na vstupních podmínkách a je nutné je kombinovat s dalšími režimy činností. Například se jedná o klasické režimy činnosti CBC, OFB, CTR, XTS nebo tzv. moderní režimy činnosti CCM, EAX, GCM, SIV a OCB, které ovšem mají fixní strukturu a nejsou uzpůsobeny potřebám šifry. Lze nalézt pouze omezené množství příkladů algoritmů (systémů), kde je polymorfní chování aplikované ve větší míře (viz kapitola 4). Tyto systému jsou buď teoretickým konceptem nebo nebyly nikdy využity v praxi.

Navržený a vytvořený kryptografický systém TSS v rámci disertační práce je v tomto směru unikátní. Nejen že byly jeho dílčí části navrženy, aby jejich princip byl polymorfní, ale také s důrazem, aby bylo možné všechny vytvořené části sjednotit v komplexně fungující kryptografický systém. Systém, který polymorfně odvozuje *IV*. Systém s polymorfně odvozenými parametry pro šifrování, jako substituční tabulka, počet rund, délka šifrovaných bloků. Systém, jenž samotné jádro šifrování se chová polymorfně (operace šifrování včetně pořadí operací). Poskytuje výběr z více druhů polymorfních režimů činnosti (režimu odvození klíče pro šifrování či klíčového managementu). V neposlední řadě systém využívající polymorfně a parametricky založené solení pomocí tří náhodných solí.

Většina existujících algoritmů v rámci bezpečnosti spoléhá v základu na délku klíče a s tím spojenou velikostí klíčového prostoru. V případě využití *nonce*/*IV*/tweaku/soli pro šifrování nebo režim činnosti je nutné tyto parametry vhodně sdílet, podobně jako u tajného klíče. V praxi je nutné využití asymetrické kryptografie pro jejich přenos stejně jako u klíče nebo připojení k šifrovým datům na začátek.

Navržený systém TSS nevyužívá pouze tajný klíč k zajištění bezpečnosti, ale rozšiřuje množinu klíčů, která je při délce 111 bajtů dostačující v odolání útoku hrubou silou, o uživatelsky vstup ve formě nastavení parametrů. Délka klíče 111 bajtů neboli 888 bitů byla zvolena nejen z důvodu počtu možných kombinací klíčů, ale i pro možnost odvozování parametrů a vlastností systému. Můžeme říci, že i když uživatel bude šifrovat stejným klíčem stejná otevřená data, tak při

využití odlišného uživatelské nastavení parametrů budou mít šifrová data jiný tvar. Další věc, která lze považovat za tajnou jsou vygenerované soli, protože soli k „předsolení“ a „zasolení“ jsou zašifrovány a součástí výstupních šifrových dat a sůl pro „nasolení“ *IV* je algoritmicky začleněna do šifrových dat, kdy pozice soli je variabilní, závislá na tajném klíči, uživatelském nastavení a délce otevřených dat. V porovnání se stávajícími algoritmy není nutné sdílení solí či připojení solí na začátek šifrových dat. Tato nutnost přenosu opět zůstává pro tajný klíč a tajné vstupní nastavení parametrů. Celkově se v souvislosti s aplikací solí chová systém náhodně a jeho fungování je závislé na složce *t* – čas. Protože i když budou na vstupu stejná otevřená data, stejná klíčová data pro odvození *IV*, stejné tajné uživatelské nastavení vstupních parametrů, tak sůl je vždy generována náhodně a při všech vzájemných vazbách lze tvrdit, že i výstupní šifrová data mají polymorfně náhodný tvar. V souvislosti s parametrizací, která je opět závislá i na obsahu solí, lze říci, že otevřená data jsou šifrované polymorfně náhodným způsobem. Množina způsobů šifrování je dána navrženým systémem. Vše za možnosti šifrová data zpětně dešifrovat a bez nutnosti sdílení solí.

Na základě výše uvedeného lze vidět přínos pro praxi, kdy je množina stávajících algoritmů rozšířena o nové. Zároveň je poskytnuta i potenciální možnost odolání blížícím se kvantovým počítačům → bez znalostí tajných dat, nastavení parametrů a solí nelze určit konkrétní algoritmus použitý k šifrování otevřených dat. Tato skutečnost komplikuje využití kvantových počítačů pro prolomení systému. Systém TSS poskytuje vyšší míru bezpečnosti, ale je nutné říct, že za cenu výpočetní náročnosti → rychlost při testování byla přibližně 800 krát a více pomalejší než AES (viz kapitola 8.3), proto jeho využití lze doporučit se zaměřením na šifrování menších objemů dat (cca do velikosti MB) – ochrana klíčů, hesel, kritických dokumentů apod. Dalším praktickým přínosem je potenciální využití jednotlivých části na rozšíření bezpečnosti stávajících algoritmů – využití parametrizace, polymorfního režimu činnosti nebo „obalením“ systému pomocí vytvořeného solení. Celý systém lze teoreticky využít s náhradou/změnou blokové šifry. To umožní zvýšení bezpečnosti blokových šifer – protože systém solení zvyšuje znatelně difúzi a konfúzi. Zároveň znemožňuje určení důležitých parametrů z výsledných šifrových dat – délka vstupních otevřených dat, parametry a další.

Problematika polymorfních struktur v rámci symetrické kryptografie by zasloužila větší vědeckou pozornost. Jak bylo ukázané v rámci disertační práce, tak není složité aplikovat polymorfní chování na kteroukoliv oblast symetrické kryptografie. Teoreticky by šla nejen symetrická kryptografie převést na polymorfní chování, ale například hashovací algoritmy nebo i asymetrická kryptografie. Složitější problematikou, než samotná tvorba polymorfních struktur

je jejich následné vyhodnocení a podrobná analýza. Disertační práce poskytuje solidní základy a prostor pro další bádání na poli polymorfních struktur. Zároveň práce poskytuje prostor pro podrobnější analýzu a následnou optimalizaci navržených struktur a je inspirací pro další tvorbu, aplikaci a zkoumání problematiky polymorfních struktur v kryptografii.

10 ZÁVĚR

Výzkum provedený v rámci disertační práce lze shrnout dle cílů disertační práce na

- studium a analýzu aktuálního stavu řešené problematiky polymorfních struktur v symetrické kryptografii a vymezení problematiky symetrické kryptografie v souladu s polymorfním chováním,
- výběr částí, struktur a parametrů symetrické kryptografie a návrh vhodné parametrizace a návrh polymorfizace vybraných částí,
- praktickou implementaci všech navržených struktur a propojení v komplexní kryptografický systém,
- otestování a zhodnocení vytvořených řešení.

Struktura disertační práce je sepsána v souladu s výše uvedenými cíli, kdy stěžejní část je věnována návrhu polymorfních struktur, vytvoření kryptografickému systému s názvem TSS – „Triply Salted Smallie“ a jejich otestování.

V první části disertační práce byly vymezeny základní pojmy spojené s kryptologií s důrazem na problematiku symetrickou kryptografii blokových šifer. Byly zde uvedeny základní pojmy a terminologie potřebné k vymezení a pochopení pojmů spojených s řešenou problematikou. V neposlední řadě se první část věnuje popisu základních pravidel, které jsou nutné pro pochopení problematiky návrhu algoritmů v symetrické kryptografii. Zejména vymezení pojmů difúze a konfúze nebo důležitost otevřenosti řešení.

V rámci druhé části byly krátce vymezeny pojmy a terminologie z oblasti symetrické kryptografie. Zejména jako příprava prostředků a základu pro tvorbu polymorfních struktur symetrické kryptografie, které jsou součástí vytvořeného systému TSS.

V další části disertační práce byla provedena analýza aktuálního stavu problematiky spojené se symetrickou kryptografií, kde byl kladen důraz na blokové šifry a s nimi souvisejícími algoritmy. Dále byl kladen důraz také na algoritmy vykazující polymorfní chování, kdy byly sepsány existující řešení spojená s problematikou polymorfních struktur v symetrické kryptografii. Z provedené analýzy vyplývá, že problematika polymorfních struktur je pouze okrajovou součástí kryptologie. V rámci rešerše bylo možné dohledat pouze malé množství vědeckých publikací, a i v takovém případě se nejednalo o práce, které by souvisely s ucelenou problematikou. Z těchto důvodů bylo nutné v rámci disertační práce částečně vymezit problematiku spojenou s polymorfními strukturami symetrické kryptografie. Na základě vymezení, bylo možné objevit

návrhy šifer, které mají částečně polymorfní chování → odvození substitučních tabulek či permutačních boxů na začátku či za chodu, algoritmy s proměnlivou délkou šifrovaných bloků či skupina šifer založena na tzv. tweaku (hodnotě/parametru mírně modifikující průběh šifrování). Existují i řešení či teoretické koncepty u kterých je možné vidět vyšší míru polymorfního chování → šifra DSDP, systém TurboCrypt a jeho budoucí verze (proprietární systém německé firmy), či konceptuální návrhy Z876 nebo Anigma, ale neexistuje (alespoň né veřejně známá) ucelená řešení, která by byla na polymorfní jako navržený systém TSS. Dalším důležitým zjištěním bylo, že většina algoritmů, které jsou z části polymorfní nenacházejí široké uplatnění v praxi. Výjimkou jsou například šifry Blowfish, Threefish, GOST a pár dalších. Lze jen polemizovat, proč tomu tak je. Nejpravděpodobněji tomu je v důsledku složitosti jejich analýzy, která by byla potřebná pro potvrzení a garanci potřebného stupně bezpečnosti či pro potřeby standardizace. Nicméně si zde dovolím poznamenat, že pokud je složité potvrzení kvality algoritmů, tak že neméně složité je i analyzování za účelem nalezení problémů (chyb). Což lze označit jako „dvousečnou“ zbraň a jako potenciální obavu z využívání polymorfních algoritmů. O složitosti analýzy polymorfních systému je se možné přesvědčit v rámci vytvořeného systému TSS, kdy každá část, funkce a chování souvisí s každou další částí mezi sebou a kdy je celkové chování systému odvozeno od náhodných veličin, tajných uživatelských dat a tajného vstupního nastavení parametrů uživatelem. Systém se tudíž mění na základě uživatelských vstupů tak i na základě náhodně generovaných dat, kdy celkový počet algoritmů (průběhu šifrování systému) je tak obsáhlá množina, že analýzu systému lze přirovnat k prolomení klíče hrubou silou. Jako další možný důvod, proč se polymorfní šifrovací algoritmy nevyužívají v takové míře či proč nelze dohledat dostatek vědeckých zdrojů a publikací je ten, že algoritmy možná existují, ale využívají se tajně či jako součást proprietárních systémů.

V další krátké části je vymezena základní myšlenka problematiky tvorby polymorfních struktur ve vazbě na blokové šifry. Myšlenku lze shrnout jako snahu navrhnout kryptografický algoritmus/systém, který by byl v souladu se všemi základními principy tvorby bezpečných šifrovacích algoritmů/systémů, ale kdy by před samotným šifrováním či po něm nebylo možné určit konkrétní využitý algoritmus a nastavení parametrů systému. To znamená, vytvoření a návrh kryptografického systému, který by měl koncept otevřenosti, ale který by byl vytvořen jako množina systémů, kdy využití konkrétního systému by bylo odvozeno na základě tajných vstupních dat → klíče. Tudíž by bez znalosti klíče nebylo možné znát konkrétní algoritmus, který byl využit k zašifrování otevřených dat.

Na základě této myšlenky byly zvoleny parametry, algoritmy a části problematiky kryptografie blokových šifer za cílem návrhu komplexního šifrovacího systému TSS. Systém TSS je tedy vytvořen plně v souladu s výše uvedenou myšlenkou, kdy velká část chování, funkcí včetně parametrů a průběhu šifrování jsou odvozeny od tajného uživatelského klíče. TSS lze označit jako „systém šifrovacích systémů“. Dále bylo šifrovací funkce a šifrování parametrizováno, kdy výčet parametrů lze dohledat v rámci textu disertační práce zejména v kapitolách 5.2, 7.2 a 7.4. Aktuálně je systém parametrizován na základě 19 parametrů, kdy je počítáno s budoucím možných rozšířením těchto parametrů.

V neposlední řadě byl navržený systém TSS doplněn o funkcionalitu solení → zahrnutí náhodných dat do procesu šifrování a lze říct že se celý systém včetně parametrů chová „náhodně“ v závislosti na těchto datech. Systém TSS chová nejen polymorfně, ale i polymorfně „náhodně“ → pseudo-náhodně. Proto, pokud budeme šifrovat stejná otevřená data, stejným tajným klíčem a za stejně nastavených vstupních parametrů, tak výstupem budou rozdílná šifrová data. A to při každém spuštění šifrování. V souvislosti s tím dojde i ke změně vnitřních funkcí a parametrů systému, takže lze říci, že při každém šifrování jsou otevřená data šifrované jinak (jiným šifrovacím systémem).

Další část disertační práce je tedy věnována konkrétnímu návrhu jednotlivých částí systému TSS a popisu tvorby a implementace těchto řešení. Jsou zde popsány návrhy polymorfního odvození klíče z tajných dat, tvorba polymorfních režimů činnosti blokových šifer a správa klíčů (vytvořena skupina režimů činností PM), polymorfní šifrovací jádro → bloková šifra, polymorfní parametrizace systému a vlastností a polymorfní solení. Všechny součásti systému TSS byly navrženy polymorfně.

V poslední části disertační práce je popsán průběh testování systému TSS. Z testování lze vidět, že i při vyřazení funkcionality šifrování, při použité stejného klíče a jen na základě změny nastavení parametrů se systém chová rozdílně a stejná vstupní otevřená data jsou zašifrována rozdílně → výstupem jsou rozdílná šifrová data. Dále bylo ukázáno, že systém disponuje vysokou mírou difúze (změny i beze změny otevřených dat), konfúze a pravděpodobnostní distribuce znaků (výsledná entropie šifrových dat) byla blížící se maximální hodnotě. To i v případě, že byla šifrována otevřená data reprezentující hodnoty nulových bajtů. Z testovaných vytvořených variant režimů činností v rámci skupiny PM nejlepších výsledků pravděpodobnostní distribuce výstupních znaků v klíči dosahovaly varianty číslo čtyři a pět, kdy byla výsledná distribuce rovnoměrná jedna ku dvě stě padesáti šesti.

Výstupem práce je ucelený komplexní kryptografický polymorfní šifrovací systém TSS jako demonstrace polymorfních struktur v symetrické kryptografii. Dalším přínosem práce jsou vytvořené jednotlivé návrhy polymorfních struktur, které je možné využít jako podklad pro budoucí výzkum. Je důležité poznamenat, že vytvořený systém TSS, i když vykazuje velmi dobré výsledky, je pouze konceptuálním řešením a nelze jej považovat za plnohodnotně bezpečné a aplikovatelné řešení do praxe. Pro využití systému v praxi by bylo potřebné jej podrobit podrobnější analýze, včetně rozsáhlejšímu testování.

SEZNAM POUŽITÉ LITERATURY

[1] USA. ADVANCED ENCRYPTION STANDARD (AES): Federal Information Processing Standards Publication 197. In: USA: National Institute of Standards and Technology (NIST), 2001, ročník 1996, číslo 104-106. Dostupné také z: https://nvlpubs.nist.gov/nistpubs/fips/nist.fips.197.pdf

[2] Camellia: A 128-Bit Block Cipher Suitable for Multiple Platforms [online]. 2000, 41, [cit. 2018-05-08]. Dostupné z: https://www.cosic.esat.kuleuven.be/nessie/updatedPhase2Specs/camellia/camellia-v2.pdf

[3] GOST R 34.12-2015: Block Cipher "Kuznyechik" [online]. 2016 [cit. 2018-05-08]. ISSN 2070-1721. Dostupné z: https://tools.ietf.org/html/rfc7801

[4] BAI, K.C.Shyamal, Dr.M.V. SATYANARAYANA a Dr. P.A. VIJAYA. Variable Size Block Encryption using Dynamic-key Mechanism (VBEDM). International Journal of Computer Applications [online]. 2011, August 2011, 27(7) [cit. 2018-05-08]. ISSN 0975 – 8887. Dostupné z: https://www.ijcaonline.org/volume27/number7/pxc3874539.pdf

[5] SCHROEPPEL, R. The Hasty Pudding Cipher, submission to NIST AES competition, 1998.

[6] LISKOV, Moses, Ronald L. RIVEST a David WAGNER. Tweakable Block Ciphers. Journal of Cryptology [online]. USA, 2011, August 2011, 24(3), 588-613 [cit. 2018-05-08]. DOI: 10.1007/s00145-010-9073-y. ISSN 0933-2790. Dostupné z: http://link.springer.com/10.1007/s00145-010-9073-y

[7] JEAN, Jérémy, Ivica NIKOLIĆ a Thomas PEYRIN. Tweaks and Keys for Block Ciphers: The TWEAKEY Framework. Journal of Cryptology [online]. USA, 2011, 2014, 24(3), 588-613 [cit. 2018-05-08]. DOI: 10.1007/978-3-662-45608-8_15. ISBN 10.1007/978-3-662-45608-8_15. ISSN 0933-2790. Dostupné z: http://link.springer.com/10.1007/978-3-662-45608-8_15

[8] BEIERLE, Christof, Jérémy JEAN, Stefan KÖLBL, et al. The SKINNY Family of Block Ciphers and Its Low-Latency Variant MANTIS: The TWEAKEY Framework. Journal of Cryptology [online]. USA, 2011, 2016-07-21, 24(3), 588-613 [cit. 2018-05-08]. DOI: 10.1007/978-3-662-53008-5_5. ISBN 10.1007/978-3-662-53008-5_5. ISSN 0933-2790. Dostupné z: http://link.springer.com/10.1007/978-3-662-53008-5_5

[9] CHEN, Like, Runtong ZHANG, Stefan KÖLBL, et al. A Key-dependent Cipher DSDP: The TWEAKEY Framework. Journal of Cryptology [online]. USA, 2011, 2008, 24(3), 588-613 [cit. 2018-05-08]. DOI: 10.1109/ISECS.2008.77. ISBN 10.1109/ISECS.2008.77. ISSN 0933-2790. Dostupné z: http://ieeexplore.ieee.org/document/4606078/

[10] SCHNEIER, Bruce, Runtong ZHANG, Stefan KÖLBL, et al. Description of a new variable-length key, 64-bit block cipher (Blowfish): The TWEAKEY Framework. Journal of Cryptology [online]. USA, 2011, 1994-6-8, 24(3), 588-613 [cit. 2018-05-08]. DOI: 10.1007/3-540-58108-1_24. ISBN 10.1007/3-540-58108-1_24. ISSN 0933-2790. Dostupné z: http://link.springer.com/10.1007/3-540-58108-1_24

[11] ŽÁČEK, Petr. Návrh nové symetrické šifry pro mobilní zařízení. Zlín, 2014. Diplomová práce. Univerzita Tomáše Bati ve Zlíně. Vedoucí práce Ing. David Malaník, Ph. D.

[12] ŠENKEŘÍK, Roman. Přednášky z předmětu Kryptologie. 2005-2013.

[13] Current Modes. In: Special Publication 800-38A: First Part: Five Confidentiality Modes 2001. [Online]. Available from: http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf

[14] Modes Development. In: National Institute of Standards and Technology: Computer Security Resource Center [online]. 2001, http://csrc.nist.gov/groups/ST/toolkit/BCM/modes_development.html

[15] PATERSON, Kenneth G. a Arnold YAU. Padding Oracle Attacks on the ISO CBC Mode Encryption Standard. OKAMOTO, Tatsuaki, ed. Topics in Cryptology – CT-RSA 2004 [online]. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, 2004, s. 305-323 [cit. 2018-05-08]. Lecture Notes in Computer Science. DOI: 10.1007/978-3-540-24660-2_24. ISBN 978-3-540-20996-6. Dostupné z: http://link.springer.com/10.1007/978-3-540-24660-2_24

[16] https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-67r2.pdf

[17] SHANNON, Claude Elwood. COVER SHEET FOR TECHNICAL MEMORANDA
RESEARCH DEPARTMENT - SUBJECT: A mathematical theory of cryptography - case 20878 (U). , 1945.

[18] BIRYUKOV, Alex, Daniel DINU a Dmitry KHOVRATOVICH, 2017. Argon2: the memory-hard function for password hashing and other applications.

University of Luxembourg. Dostupné také z: https://www.cryptolux.org/index.php/Argon2

[19] HARKINS, D., 2017. Adding Support for Salted Password Databases to EAP-pwd. Internet Engineering Task Force (IETF). Dostupné také z: https://tools.ietf.org/html/rfc8146

[20] JOSEFSSON, S. a C. PERCIVAL, 2016. The scrypt Password-Based Key Derivation Function. Internet Engineering Task Force (IETF). Dostupné také z: https://tools.ietf.org/html/rfc7914

[21] MORIARTY, K., B. KALISKI a A. RUSCH, 2017. PKCS #5: Password-Based Cryptography Specification Version 2.1. Internet Engineering Task Force (IETF). Dostupné také z: https://tools.ietf.org/html/rfc8018

[22] Linux on IBM Systems: ANSI X9.23 cipher block chaining, 2016. IBM Documentation [online]. IBM Corporation [cit. 2021-5-2]. Dostupné z: https://www.ibm.com/docs/en/linux-on-systems?topic=processes-ansi-x923-cipher-block-chaining

[23] ČSN ISO 10126-1, 1996. BANKOVNICTVÍ - Postupy pro šifrování zpráv (bankovní služby pro velkou klientelu): Část 1: Obecné zásady. Hradec Králové: Český normalizační institut - TECHNOR.

[24] KALISKI, B., 1998. PKCS #7: Cryptographic Message Syntax Version 1.5. Internet Engineering Task Force (IETF). Dostupné také z: https://tools.ietf.org/html/rfc2315

[25] Simon and Speck: Block Ciphers for the Internet of Things. Information Technology Laboratory: COMPUTER SECURITY RESOURCE CENTER [online]. HEADQUARTERS 100 Bureau Drive Gaithersburg, MD 20899: NIST: National Institute of Standards and Technology [cit. 2021-5-2]. Dostupné z: https://csrc.nist.gov/csrc/media/events/lightweight-cryptography-workshop-2015/documents/papers/session1-shors-paper.pdf

[26] NEEDHAM, Roger M. a David J. WHEELER, 1997. Tea extensions. Dostupné také z: http://www.cix.co.uk/~klockstone/xtea.pdf

[27] WHEELER, David J., 1995. TEA, a Tiny Encryption Algorithm. Lecture Notes in Compure Science [online]. Springer, FSE 1994, 363-366 [cit. 2021-5-2]. Dostupné z: https://link.springer.com/content/pdf/10.1007/3-540-60590-8_29.pdf

[28] MERKLE, Ralph C. Fast software encryption functions [online]. . . Alfred J. MENEZES a Scott A. VANSTONEBerlin, Heidelberg: Springer Berlin

Heidelberg, 1991. 477-501 s. ID: 10.1007/3-540-38424-3_34. ISBN 9783-540384243.

[29] DOLMATOV, V., 2010. GOST 28147-89: Encryption, Decryption, and Message Authentication Code (MAC) Algorithms. Internet Engineering Task Force (IETF). Dostupné také z: https://tools.ietf.org/html/rfc5830

[30] ALTIGANI, Abdelrahman, Shafaatunnur HASAN a Bazara BARRY. THE NEED FOR POLYMORPHIC ENCRYPTION ALGORITHMS: A REVIEW PAPER. Journal of Theoretical and Applied Information Technology [online]. 2020, vol. 98, s. 360-377.

[31] BIRYUKOV, Alex a Dmitry KHOVRATOVICH. Related-key cryptanalysis of the full AES-192 and AES-256 [online]. . . Mitsuru MATSUIBerlin, Heidelberg: Springer Berlin Heidelberg, 2009. 1-18 s. ID: 10.1007/978-3-642-10366-7_1. ISBN 9783-642103667.

[32] SAEB, Magdy. The Chameleon Cipher-192 (CC-192): A polymorphic cipher. Secrypt2009 [online]. 2009, vol. SECRYPT2009 Proceedings.

[33] KALYAN DUTTA, I. et al. Lightweight polymorphic encryption for the data associated with constrained internet of things devices [online]. , 2020. 1-6 s.

[34] ALTIGANI, A. et al. A Polymorphic Advanced Encryption Standard – A Novel Approach. IEEE Access [online]. 2021, vol. 9, s. 20191-20207. ISSN 2169-3536.

[35] TrueCrypt, 2014. TrueCrypt [online]. [cit. 2021-5-2]. Dostupné z: http://truecrypt.sourceforge.net

[36] VeraCrypt: Documentation, 2021. VeraCrypt [online]. [cit. 2021-5-2]. Dostupné z: https://veracrypt.fr/en/Documentation.html

[37] TurboCrypt User Manual, 2018. Ciphers.de [online]. Schoeffengrund, Germany [cit. 2021-5-2]. Dostupné z: http://www.turbocrypt.com/documents/whitepapers/turbocrypt_user_manual.pdf

[38] ROELGEN, C. B., 2010. Approaching the Perfect Cipher by Trespassing the block size confinement: The Polymorphic Giant Block Encryption Algorithm. Dostupné také z: http://www.turbocrypt.com/documents/whitepapers/giant_block_size_polymorphic_cipher.pdf

[39] Polymorphic Encryption. Ciphers.de: TurboCrypt [online]. Schoeffengrund, Germany [cit. 2021-5-2]. Dostupné z:

http://www.turbocrypt.com/pages/content/page.aspx?contcatid=6573f20a8e1e4ce5a552553b59002766

[40] KNUDSEN, Lars R. a Matthew J. B. ROBSHAW, [2011]. The block cipher companion. Heidelberg: Springer. ISBN 978-3-642-17341-7.

[41] Symmetric ciphers: Modern modes of operation, 2020. PyCryptodome - documentation: Revision ca247079 [online]. [cit. 2021-5-2]. Dostupné z: https://pycryptodome.readthedocs.io/en/latest/src/cipher/modern.html

[42] MORRIS, J. Dworkin. 2004. SP 800-38C. Recommendation for Block Cipher Modes of Operation: the CCM Mode for Authentication and Confidentiality. Technical Report. National Institute of Standards & Technology, Gaithersburg, MD, USA. [online]. Dostupné z: https://doi.org/10.6028/NIST.SP.800-38C

[43] BELLARE, Mihir, Phillip ROGAWAY a D. WAGNER. EAX: A Conventional Authenticated-Encryption Mode. IACR Cryptology ePrint Archive [online]. 2003, vol. 2003, s. 69. Dostupné z: https://csrc.nist.gov/csrc/media/projects/block-cipher-techniques/documents/bcm/proposed-modes/eax/eax-spec.pdf

[44] MORRIS, J. Dworkin. 2007. SP 800-38D. Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC. National Institute of Standards & Technology, Gaithersburg, MD, USA. [online]. Dostupné z: https://doi.org/10.6028/NIST.SP.800-38D

[45] ROGAWAY, Phillip, 2007. The SIV Mode of Operation for Deterministic Authenticated-Encryption (Key Wrap) and Misuse-Resistant Nonce-Based Authenticated-Encryption. Draft 0.32. Thomas Shrimpton - Portland State University. Dostupné také z: https://web.cs.ucdavis.edu/~rogaway/papers/siv.pdf

[46] ROGAWAY, Phillip, Mihir BELLARE, John BLACK a Ted Krovetz. 2001. OCB: A Block-Cipher Mode of Operation for Efficient Authenticated Encryption. National Institute of Standards & Technology, Gaithersburg, MD, USA. [online]. Dostupné z: https://csrc.nist.gov/CSRC/media/Projects/Block-Cipher-Techniques/documents/BCM/proposed-modes/ocb/ocb-spec.pdf

[47] Summarizing Known Attacks on Transport Layer Security (TLS) and Datagram TLS (DTLS), 2015. Internet Engineering Task Force (IETF). Dostupné také z: https://tools.ietf.org/html/rfc7457

[48] Transport Layer Security, 2001-. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation [cit. 2021-5-2]. Dostupné z: https://en.wikipedia.org/wiki/Transport_Layer_Security

[49] Kostadin Bajalcaliev Cryptography Page [online], 1995. Makedonie [cit. 2021-5-3]. Dostupné z: https://kbajalc.tripod.com

[50] MORRIS, J. Dworkin. 2010. SP 800-38E. Recommendation for Block Cipher Modes of Operation: the XTS-AES Mode for Confidentiality on Storage Devices. National Institute of Standards & Technology, Gaithersburg, MD, USA. [online]. Dostupné z: https://doi.org/10.6028/NIST.SP.800-38E

[51] https://docs.python.org/3/library/os.html

SEZNAM POUŽITÝCH ZKRATEK

AEAD	Authenticated Encryption with Associated Data
AES	Advanced Encryption Standard
ANSI	American National Standards Institute
BEAST	Browser Exploit Against SSL/TLS
CBC	Cipherblock Chaining
CCM	Counter with CBC-MAC
CFB	Cipher Feedback
CPRNG	Chaotic Pseudo-Random Number Generator
CRIME	Compression Ration Info-leak Made Easy
CRYPTREC	Cryptography Research and Evaluation Committees
DC	Deterministic Chaos
DSDP	Dependent S-Box Dependent P-Box
ECB	Electronic Codebook
GCM	Galois/Counter mode
GOST	Gosudarstvenii Standart
HKDF	HMAC-based Extract-and-Expand Key Derivation Function
HW	Hardware
IDEA	International Data Encryption Standard
ISO	International Organization for Standartization
IV	Inicializační Vektor
LM	Logistic Maps
LUKS	Linux Unified Key Setup
NESSIE	New European Schemes for Signatures, Integrity and Encryption

NIST	National Institute of Standards and Technology
NSA	National Security Agency
OCB	Offset CodeBook mode
OFB	Output Feedback
OT	Otevřený text (otevřená data)
PBKDF2	Password based key derivation function 2
PKCS	Public Key Cryptographic Standards
PM	Polymorphous Modes
POODLE	Padding Oracle on Downgraded Legacy Encryption
PRNG	Pseudo-Random Number Generator
SIV	Synthetic Initialization Vector
TDES/TDEA	Triple Data Encryption Standard / Triple Data Encryption Algorithm
TEA	Tiny Encryption Algorithm
TLS	Transport Layer Security
TSS	Triply Salted Smallie
WEP	Wired Equivalent Privacy
WPA	Wi-Fi Protected Access
XEX	XOR-encrypt-XOR
XOR	eXclusive OR
XTEA	eXtended Tiny Encryption Algorithm
XTS	XEX-based tweaked-codebook mode with ciphertext stealing

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

PUBLIKAČNÍ ČINNOST AUTORA

Mezinárodní patentová přihláška

Jašek R. [25], Oulehla M. [35], Žáček P. [6], Krňávek J. [14], Lázecký V. [5], Makowski J. [5], Malík T. [5], Malík J. [5] Identity and License Verification System for Working with Highly Sensitive Data

Konference a časopisy

[A.1] ŽÁČEK, Petr, JAŠEK, Roman, MALANÍK, David. Using the Deterministic Chaos in Variable Mode of Operation of Block Ciphers. In *Artificial Intelligence Perspectives and Applications*. Heidelberg: Springer-Verlag Berlin, 2015, s. 347-354. ISSN 2194-5357. ISBN 978-3-319-18475-3.

[A.2] ŽÁČEK, Petr, JAŠEK, Roman, MALANÍK, David. Group of the Polymorphous Modes of Operation - PM. In *Proceedings of the 2016 Future Technologies Conference (FTC)*. New Jersey, Piscataway: IEEE, 2016, s. 1314-1315. ISBN 978-1-5090-4171-8.

[A.3] ŽÁČEK, Petr, JAŠEK, Roman, MALANÍK, David. Improvement of CPRNG of the PM-DC-LM Mode and Comparison with its Previous Version. In *Tenth International Conference on Emerging Security Information, Systems and Technologies*. Wilmington: IARIA XPS Press, 2016, s. 57-62. ISBN 978-1-61208-493-0.

[A.4] ŽÁČEK, Petr, JAŠEK, Roman, KRÁLÍK, Lukáš, MALANÍK, David, HOLBÍKOVÁ, Petra. Analysis of the chaotic pseudo-random generator of the PM-DC-LM mode based on the position of the returned numbers. In *2017 International Conference on Logistics, Informatics and Service Sciences (LISS)*. New Jersey, Piscataway: IEEE, 2017, s. nestrankovano. ISBN 978-1-5386-1047-3.

[A.5] ŽÁČEK, Petr, JAŠEK, Roman, MALANÍK, David. A Comparison of the PM-DC-LM Mode With Other Common Operational Block Cipher Modes. In *The Ninth International conference on Emerging Security Information, Systems and Technologies*. Wilmington: IARIA, 2015, s. 44-48. ISSN 2162-2116. ISBN 978-1-61208-427-5.

[A.6] ŽÁČEK, Petr, JAŠEK, Roman, MALANÍK, David. Possibilities and Testing of CPRNG in Block Cipher Mode of Operation PM-DC-LM. In *Proceedings of PPS-30: The 30th International Conference of the Polymer Processing Society*. Melville: American Institute of Physics Publising Inc., 2016, s. "nestrankovano". ISSN 0094-243X. ISBN 978-0-7354-1309-2.

[A.7] ŽÁČEK, Petr, JAŠEK, Roman, MALANÍK, David, KRÁLÍK, Lukáš, HOLBÍKOVÁ, Petra. Using the SHA-3 to Derive Encryption Keys Based on Key-file. *Proceedings - 2018 2nd European Conference on Electrical Engineering and Computer Science, EECS 2018*. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers Inc., 2019, s. 348-351. ISBN 978-1-72811-929-8.

ODBORNÝ ŽIVOTOPIS AUTORA

OSOBNÍ ÚDAJE Ing. Petr Žáček

Kříby 4713, 76005 Zlín (Česká republika)

+420 734 304 234

PRACOVNÍ ZKUŠENOSTI

01/10/2017–do současnosti Asistent

Univerzita Tomáše Bati ve Zlíně, Zlín (Česká republika)

1. Výuka předmětů
   - Bezpečnost informačních systémů, Kryptologie, Testování software, Programování
2. Člen laboratoře pro penetrační testování PTLab
   - Penetrační testování sítí, webových stránek
   - Konzultační činnost v oblasti kybernetické bezpečnosti
   - Etický hacking

VZDĚLÁNÍ, ODBORNÁ PŘÍPRAVA A KURZY

2012–2014 Vysokoškolské vzdělání II. stupně

Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky; Bezpečnostní technologie, systémy a management – technické zaměření, Zlín (Česká republika), Zlín (Česká republika)

Diplomová práce – Návrh nové symetrické šifry pro mobilní zařízení v jazyce Python 3.x

2014–do současnosti Vysokoškolské vzdělání III. stupně

Univerzita Tomáše Bati ve Zlíně, Fakulta aplikované informatiky; Inženýrská informatika, Zlín (Česká republika), Zlín (Česká republika)

OSOBNÍ DOVEDNOSTI

Mateřský jazyk Čeština

Další jazyky

	POROZUMĚNÍ		MLUVENÍ		PÍSEMNÝ PROJEV
	Poslech	Čtení	Ústní interakce	Samostatný ústní projev	
Angličtina	B2	B2	B1	B2	B2

Úrovně: A1 a A2: základní uživatel - B1 a B2: samostatný uživatel - C1 a C2: zkušený uživatel
Společný evropský referenční rámec pro jazyky

Certifikáty

The Complete Ethical Hacking Course: Beginner to Advanced! - UC-5W8MTNLZ
Programming for Everybody (Python)
Cryptography I.
ISTQB - Certified Tester Foundation Level - 00153/15
ISTQB - Certified Tester Foundation Level - Agile Tester Extension - 00013/16

DOPLŇUJÍCÍ INFORMACE

Projekty

IGA/FAI/2015/047 - Variabilní struktura v režimu činnosti blokových šifer (Hlavní řešitel)

IGA/FAI/2016/028 - Rozšíření a možnosti vylepšení polymorfních režimů činnosti blokových šifer ze skupiny PM (Hlavní řešitel)

IGA/CebiaTech/2017/007 - Softwarová podpora pro školení a testování správců IT (Spoluřešitel)

IGA/CebiaTech/2018/007 - Softwarová podpora pro školení a testování správců IT- II. (Hlavní řešitel)

FAI2A/2019 - Kvantový generátor náhodných čísel ve výuce předmětu Kryptologie (A3KRY) (Spoluřešitel)

RVO/CEBIA/2018/001 - Aplikace inženýrské informatiky (Spoluřešitel)

RVO/CEBIA/2019/001 - Aplikace inženýrské informatiky (Spoluřešitel)

CZ.01.1.02/0.0/0.0/15_019/0004580 – Platforma INFOS (Spoluřešitel)

CZ.02.2.69/0.0/0.0/16_015/0002204 – Strategický project UTB ve Zlíně (Spoluřešitel)

CZ.01.1.02/0.0/0.0/17_107/0012503 - Výzkum a vývoj eHealth Integrované aplikační platformy Telemedicíny (Spoluřešitel)

CZ.02.2.69/0.0/0.0/18_056/0012951 - DUO UTB: Strategický projekt UTB ve Zlíně II. (Spoluřešitel)

Penetrační testování pro soukromý sektor – roky 2016 až současnost (Spoluřešitel)