Slovenská Technická Univerzita v Bratislave
Fakulta Elektrotechniky a Informatiky

Evidenčné číslo: FEI-12306-36029

Kryptoanalýza prúdových šifier

Dizertačná práca

Študijný program:	Aplikovaná informatika
Číslo študijného odboru:	2511
Názov študijného odboru:	9.2.9 aplikovaná informatika
Školiace pracovisko:	Ústav informatiky a matematiky
Školiteľ:	doc. Ing. Milan Vojvoda, PhD.
Konzultant:	prof. RNDr. Peter Horák, DrSc.

Bratislava 2014 **Ing. Viliam Hromada**

Slovenská technická univerzita v Bratislave
Ústav informatiky a matematiky

Fakulta elektrotechniky a informatiky
Akademický rok: 2014/2015
Evidenčné číslo: FEI-12306-36029

ZADANIE DIZERTAČNEJ PRÁCE

Študent:	**Ing. Viliam Hromada**
ID študenta:	36029
Študijný program:	Aplikovaná informatika
Študijný odbor:	9.2.9 aplikovaná informatika
Vedúci práce:	doc. Ing. Milan Vojvoda, PhD.
Konzultant:	prof. RNDr. Peter Horák, DrSc.

Názov práce: **Kryptoanalýza prúdových šifier**

Špecifikácia zadania:

Témou dizertačnej práce je kryptoanalýza prúdových šifier, resp. analýza vybraných techník priamych a nepriamych útokov. Cieľom práce je štúdium využitia chybovej analýzy pri útokoch na prúdové šifrátory a návrh nových prúdových šifier.
Ďalej je cieľom dizertačnej práce štúdium využitia chybovej analýzy pri útoku na prúdovú šifru Trivium.
Literatúra:
1) J. Menezes, P. C. van Oorschot, S. A. Vanstone: Handbook of Applied Cryptography, CRC-Press, Boca Raton, 1996.
2) J. J. Hoch, A. Shamir: Fault Analysis of Stream Ciphers, Proceedings of Cryptographic Hardware and Embedded Systems – CHES 2004, LNCS 3156, Springer-Verlag, 2004, pp. 240-253.
3) Časopisecká literatúra podľa odporučenia vedúceho práce.

Riešenie zadania práce od:	05. 09. 2011
Dátum odovzdania práce:	05. 09. 2014

L. S.

Ing. Viliam Hromada
riešiteľ

prof. RNDr. Otokar Grošek, PhD.
vedúci pracoviska

prof. RNDr. Otokar Grošek, PhD.
garant študijného programu

ČESTNÉ VYHLÁSENIE

Čestne vyhlasujem, že som túto dizertačnú prácu vypracoval sám s použitím uvedenej literatúry, s výnimkou článkov, kde je spoluautorstvo explicitne uvedené.

V Bratislave, 30. 6. 2014

.....
Viliam Hromada

Chcem sa poďakovať prof. RNDr. Otokarovi Grošekovi, PhD., prof. RNDr. Petrovi Horákovi, DrSc. a doc. Ing. Milanovi Vojvodovi, PhD. za cenné rady, pripomienky, ochotu, trpezlivosť a obetovaný čas pri vzniku tejto dizertačnej práce a počas mojich doktorandských štúdií. Taktiež sa chcem poďakovať kolektívu Ústavu informatiky a matematiky FEI STU za vytvorenie príjemnej pracovnej a tvorivej atmosféry.

V neposlednom rade ďakujem mojej rodine a priateľke Martine za podporu počas celého štúdia i mimo neho.

Obsah

Zoznam obrázkov

Zoznam tabuliek

Kapitola 1

Úvod

Ľudská komunikácia je stará ako ľudstvo samo. Už z obdobia pred našim letopočtom sú známe prípady, keď si chceli dvaja korešpondenti vymeniť správu bez toho, aby ju dokázal čítať niekto iný, t.j. chceli túto komunikáciu nejakým spôsobom alebo utajiť, alebo správu previesť to takej formy, aby bola pre iného čitateľa nečitateľná.

Myšlienka utajenia správy, či jej transformácie do nečitateľnej podoby sa zachovala dodnes. V dnešnej dobe je bezpečnosť komunikácie jednou z priorít modernej spoločnosti, bez ktorej by nefungovala prakticky žiadna sféra verejného či súkromného života. Ochrana správ či dát je však len jednou z oblastí informačnej bezpečnosti. Medzi ďalšie patria zabezpečenie integrity dát (t.j. uistenie, že nedošlo k nedovolenej zmene dát), identifikácia/autentifikácia osoby, ktorá k dátam pristupuje (t.j. uistenie, že k dátam má prístup len povolaná osoba) či zabezpečenie dostupnosti dát (t.j. povolaná osoba musí mať vždy možnosť mať dáta k dispozícii).

Na zabezpečenie utajenia obsahu správy sa v dnešnej dobe používa šifrovanie, respektíve šifrovacie algoritmy - nazývané aj kryptosystémy. Zjednodušene sa dá povedať, že kryptosystém slúži na to, aby správu z jej čitateľnej podoby (takáto správa sa nazýva otvorený text - OT) transformoval do jej nečitateľnej podoby (takáto správa sa nazýva zašifrovaný text - ZT) pomocou tajného parametra - šifrovacieho kľúča. Takémuto procesu hovoríme šifrovanie. Samozrejme, kryptosystémy musia zabezpečiť aj proces opačný, t.j. transformáciu správy z jej nečitateľnej podoby späť na čitateľnú podobu, aby ju mohol legitímny príjemca, majiteľ dešifrovacieho kľúča, bez problémov prečítať. Tomuto procesu hovoríme dešifrovanie.

Dnešné kryptosystémy delíme podľa toho, ako šifrujú ten istý blok textu, na blokové šifry a prúdové šifry. Zatiaľ, čo blokové šifry zašifrujú vždy ten istý otvorený text, resp. jeho časť, pri použití rovnakého kľúča na ten istý zašifrovaný text (v základnom režime ECB - angl. Electronic Code-Book), pri prúdových šifrách to neplatí, t.j. rovnaké bloky otvoreného textu zašifruje prúdová šifra na iné bloky zašifrovaného textu.

Z hľadiska použitých kľúčov, resp. kľúča, delíme kryptosystémy na symetrické a

asymetrické (nazývané aj kryptosystémy s verejným kľúčom). Symetrické kryptosystémy vyžadujú, aby si korešpondenti (Alica a Bob) pred komunikáciou vymenili tajný kľúč K (pre oboch rovnaký), pomocou ktorého budú šifrovať/dešifrovať svoju komunikáciu, pričom jeho výmena prebieha cez zabezpečený kanál a nie je súčasťou šifrovacieho algoritmu. Pri kryptosystémoch s verejným kľúčom zverejnia Alica a Bob svoje šifrovacie kľúče E_A, E_B (tzv. verejné kľúče). Ak teraz chce Alica komunikovať s Bobom, vyhľadá si jeho verejný kľúč E_B, pomocou ktorého zašifruje správu a odošle ju Bobovi. Ten si ju spätne dešifruje pomocou svojho tzv. súkromného kľúča D_B (obdobne aj naopak, Bob zašifruje správu pomocou Alicinho verejného kľúča E_A, odošle ju a Alica si správu spätne dešifruje pomocou svojho súkromného kľúča D_A). Bezpečnosť tohoto systému je založená na tom, že nik iný okrem oprávneného prijímateľa nepozná jeho súkromný kľúč a nie je schopný ho v reálnom čase zo známeho verejného kľúča určiť, resp. vypočítať (t.j. len Alica pozná D_A) a útočník (nazývaný aj Oskar), ho nie je schopný v reálnom čase vypočítať zo známeho E_A.

Formálna definícia kryptosystému je nasledovná:

Definícia 1 *Kryptosystém je pätica* $(\mathcal{P}, \mathcal{C}, \mathcal{K}, \mathcal{E}, \mathcal{D})$*, kde*

- $\mathcal{P}$ *je množina otvorených textov napísaných s pomocou znakov abecedy otvorených textov* $\mathcal{A}_{OT}$*, t.j.* $\mathcal{P} = \mathcal{A}_{OT}^*$*,*
- $\mathcal{C}$ *je množina zašifrovaných textov napísaných s pomocou znakov abecedy zašifrovaných textov* $\mathcal{A}_{ZT}$*, t.j.* $\mathcal{C} = \mathcal{A}_{ZT}^*$*,*
- $\mathcal{K}$ *je množina kľúčov,*
- $\mathcal{E}$ *je množina šifrovacích transformácií* $E_e : \mathcal{P} \to \mathcal{C}$*, parametrizovaných parametrom* $e \in \mathcal{K}$*,*
- $\mathcal{D}$ *je množina dešifrovacích transformácií* $D_d : \mathcal{C} \to \mathcal{P}$*, parametrizovaných parametrom* $d \in \mathcal{K}$*, pričom platí* $D_d(E_e(m)) = m$*,* $\forall m \in \mathcal{P}$*.*

Útoky na kryptosystémy možno deliť na dve triedy: na priame útoky a na nepriame útoky. Priame útoky sú zamerané na algoritmickú podstatu kryptosystému, bez ohľadu na jeho implementáciu. Nepriame útoky využívajú fyzickú implementáciu kryptosystému a zahŕňajú širokú paletu techník, ktoré alebo poskytujú útočníkovi nejakú „vnútornú“ informáciu o procese šifrovania (ako napríklad časová analýza - angl. time analysis [32] alebo napäťová analýza - angl. power analysis [33]), alebo mu dovoľujú tento proces ovplyvniť (preklápanie bitov v pamäti zariadenia pomocou žiarenia, atď.). Chybová analýza študuje, aký efekt majú jednotlivé indukované chyby na zašifrovaný text, s cieľom získať aspoň čiastočnú informáciu alebo o kľúči, alebo o vnútornom stave šifrovacieho zariadenia.

1.1 Ciele dizertačnej práce

Táto dizertačná práca sa zaoberá prúdovými šiframi z dvoch hľadísk: návrhu nových prúdových šifier a vykonania chybovej analýzy vybraných prúdových šifier. Vedeckými cieľmi, resp. tézami, tejto dizertačnej práce boli nasledovné body:

1. Prispieť k chybovej analýze prúdových šifier analýzou niektorého z finalistov projektu eSTREAM pomocou techniky, ktorá ešte nebola na tohto finalistu aplikovaná.

2. Prispieť k oblasti návrhu prúdových šifier návrhom novej prúdovej šifry, resp. nového generátora náhodných čísiel.

Jadro tejto dizertačnej práce tvoria štyri články. Dva články sú zamerané na chybovú analýzu prúdových šifier a dva články sa zaoberajú návrhom nových prúdových šifier, resp. generátorov náhodných čísiel.

V prvom článku s názvom „Útok fázovým posunom na prúdovú šifru Trivium“ sa zaoberáme analýzou prúdovej šifry Trivium pomocou fázového posunu (1. téza). Použitie inej techniky chybovej analýzy ako štandardné preklápanie bitov v registroch má za následok zníženie dátovej zložitosti útoku v porovnaní s inými útokmi dostupnými v literatúre o 70%. Táto práca bola zároveň prezentovaná na konferencii *Central European Conference on Cryptology - CECC 2014* v Budapešti v máji 2014. Článok bol spoločnou prácou s Ing. Jurajom Vargom z Ústavu informatiky a matematiky FEI STU.

Druhý článok s názvom „Chybová analýza prúdovej šifry LILI-128“ je ďalším príspevkom k chybovej analýze prúdových šifier. Pomocou techniky fázového posunu sa nám opäť podarilo vylepšiť pôvodný útok na túto prúdovú šifru a zmenšiť počet chýb potrebných na nájdenie počiatočného naplnenia šifry. Tento článok vychádza z diplomovej práce autora tejto dizertačnej práce [26] a bol prezentovaný na konferencii *Mikulášská kryptobesídka 2011* v Prahe v decembri 2011. Článok bol spoločnou prácou s doc. Ing. Milanom Vojvodom, PhD. z Ústavu informatiky a matematiky FEI STU.

Tretí článok „Prúdová šifra založená na šifre Fialka M-125“ sa zaoberá návrhom prúdovej šifry založenej na sovietskom šifrátore z obdobia studenej vojny Fialka M-125 (2. téza). Šifru sme navrhli tak, aby kopírovala princíp fungovania Fialky - nepravidelne taktované rotory pracujúce v 2 nezávislých režimoch. Následne sme náš návrh podrobili štatistickým testom. Tento návrh bol prezentovaný na konferencii *Central European Conference on Cryptology - CECC 2013* v máji 2013 a príslušný článok vyšiel v časopise *Tatra Mountains Mathematical Publications*. Článok bol spoločnou prácou s Ing. Eugenom Antalom z Ústavu informatiky a matematiky FEI STU.

Štvrtý článok „Použitie kryptosystému Poly-Dragon v generátore náhodných čísiel MSTg“ sa zaoberá návrhom nového generátora náhodných čísiel, použitím krypto-

systému s verejným kľúčom Poly-Dragon v kombinácii s generátorom náhodných čísiel MSTg (2. téza). Šifra Poly-Dragon sa využije na vygenerovanie prvkov náhodného pokrytia grupy, ktoré sa následne využíva v generátore MSTg. Túto novú konštrukciu sme podrobili štatistickým testom na otestovanie miery pseudonáhodnosti výstupu. Táto práca bola prezentovaná na konferencii *International Student Conference on Applied Mathematics and Infomatics - ISCAMI 2012* v máji 2012 a príslušný článok bol prijatý na publikovanie v časopise *Tatra Mountains Mathematical Publications*. Článok bol spoločnou prácou s doc. Ing. Milanom Vojvodom, PhD z Ústavu informatiky a matematiky FEI STU.

1.2 Štruktúra dizertačnej práce

Táto dizertačná práca je koncipovaná ako komentovaný súbor publikovaných článkov. Jej jadro tvoria 4 vyššie uvedené články združené do 2 kapitol - tematických celkov. Ku každému článku sú stručne uvedené ciele a výsledky vedeckej práce.

Práca pozostáva z 5 kapitol. Prvú kapitolu tvorí úvod. Druhá kapitola sa zaoberá teoretickým úvodom do prúdových šifier a chybovej analýzy. Sú v nej predstavené základné techniky chybovej analýzy a delenie chýb z rôznych hľadísk. Jadro práce tvoria tretia a štvrtá kapitola. Tretia kapitola sa zaoberá chybovou analýzou vybraných typov prúdových šifier, konkrétne generátora LILI-128 a prúdovej šifry Trivium. Štvrtá kapitola obsahuje 2 návrhy nových prúdových šifier - prúdovej šifry založenej na sovietskom šifrátore Fialka M-125 a generátora náhodných čísiel založeného na náhodných pokrytiach konečných grúp v kombinácii s kryptosystémom Poly-Dragon. Záverečná kapitola obsahuje zhrnutie výsledkov dizertačnej práce.

Kapitola 2

Teoretický úvod

2.1 Prúdové šifry

Táto kapitola je spracovaná podľa knihy [38] a podľa diplomovej práce [26].

Ako je naznačené v úvode tejto práce, kryptosystémy delíme na 2 základné skupiny: blokové a prúdové šifry. Prúdové šifry šifrujú jednotlivé znaky otvoreného textu postupne (t.j. po symboloch/blokoch bitov, ktoré sú oveľa kratšie ako pri blokových šifrách), pričom šifrovacia transformácia sa s časom mení, vďaka vnútornému stavu prúdovej šifry (vnútorný stav šifry môže byť napríklad aktuálny obsah nejakého registra). V praxi to znamená, že dva rovnaké znaky sa vo všeobecnosti nezašifrujú na dva rovnaké znaky. Výhodou prúdových šifier je najmä rýchlosť a jednoduchosť ich hardvérovej realizácie v porovnaní s blokovými šiframi.

Prúdové šifry delíme na:

- Synchrónne - stav šifry nezávisí od otvoreného textu ani od zašifrovaného textu, závisí len od predchádzajúceho stavu.
- Samosynchronizujúce - stav šifry závisí nielen od predchádzajúceho stavu, ale aj od fixného počtu predtým zašifrovaných znakov.

Špeciálnym typom synchrónnej prúdovej šifry je binárna aditívna prúdová šifra.

Definícia 2 *Prúdový kľúč je postupnosť symbolov $z_0, z_1, ...,$ ktorá je výstupom generátora prúdového kľúča, ktorý je inicializovaný tajným kľúčom $k \in \mathcal{K}$.*

Definícia 3 *Nech $P = \{p_0, p_1, ..., p_{N-1}\} \in \mathcal{P}$ je otvorený text, $C = \{c_0, c_1, ..., c_{N-1}\} \in \mathcal{C}$ je zašifrovaný text a nech $z = \{z_0, z_1, ..., z_{N-1}\}$ je prúdový kľúč a navyše $p_i, z_i, c_i \in \mathbb{Z}_2$. Binárna aditívna prúdová šifra je prúdová šifra, v ktorej je šifrovacia transformácia daná vzťahom:*

$$c_i = p_i \oplus z_i, \qquad i = 0, ..., N-1,$$

a dešifrovacia transformácia je daná vzťahom:

$$p_i = c_i \oplus z_i, \qquad i = 0, ..., N-1.$$

Znak $\oplus$ *je znak operácie XOR, t.j. sčítania modulo 2.*

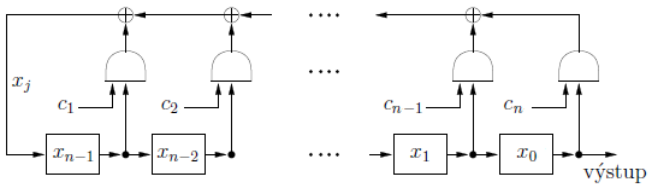
Výhodou binárnych aditívnych prúdových šifier je, že v ich konštrukcii sa využíva generátor prúdového kľúča, t.j. komponent, ktorý generuje pseudonáhodnú postupnosť bitov. Takto sa dá jednoducho zostrojiť prúdová šifra z ľubovoľného generátora náhodných bitov. Dôležitú úlohu v prúdových šifrách teda zohrávajú generátory prúdového kľúča. Navyše sa dá rovnaký hardvér použiť na šifrovanie a dešifrovanie.

Medzi základné komponenty generátorov prúdového kľúča patria tzv. lineárne spätnoväzobné registre.

Definícia 4 *Lineárny spätnoväzobný register (LFSR) dĺžky n pozostáva z vnútorného stavu X, ktorý sa skladá z n oneskorovacích prvkov* x_0, x_1, x_2, ..., x_{n-1}, *pričom každý je schopný reprezentovať jeden bit a má jeden vstup a jeden výstup; a z hodín, ktoré kontrolujú taktovanie registra. Počet bitov vnútorného stavu registra označujeme ako dĺžku registra. Počas každej časovej jednotky sa vykonajú nasledovné operácie:*

1. *Obsah bitu* x_0 *ide na výstup a tvorí časť výstupnej postupnosti.*
2. *Obsah bitu* x_i *sa posunie do bitu* x_{i-1}, *pre každé* $i, 1 \leq i \leq n-1$
3. *Nový obsah bitu* x_{n-1} *sa vypočíta ako súčet bitov z pevne danej podmnožiny prvkov* x_0, x_1, x_2, ..., x_{n-1} *modulo 2. Táto podmnožina je daná koeficientami* c_i *jednotlivých bitov; ak* $c_i = 1$, *tak príslušný bit* x_i *je súčasťou tejto podmnožiny, ak* $c_i = 0$, *tak bit* x_i *nie je súčasťou tejto podmnožiny.*

Na obrázku 2.1 je znázornená všeobecná schéma LFSR dĺžky n. Premenná c_i nadobúda hodnoty 0 alebo 1, polkruhy reprezentujú hradlo AND a bit x_j (tzv. „feedback bit“) je rovný súčtu modulo 2 tých bitov x_i $0 \leq i \leq n-1$, pre ktoré platí $c_{n-i} = 1$.



Obr. 2.1: Lineárny spätnoväzobný register dĺžky n (prevzaté z [38])

Definícia 5 *Majme lineárny spätnoväzobný register podľa obrázka 2.1. Lineárnou diferenčnou rovnicou n-tého rádu nazývame rovnicu:*

$$c_0 x_i + c_1 x_{i-1} + ... + c_{n-1} x_{i-n+1} + c_n x_{i-n} = 0,$$

kde $c_0 \neq 0, c_n \neq 0$*;* $c_i, x_i \in \{0, 1\}$.

Definícia 6 *LFSR sa vo všeobecnosti označuje* $\langle n, C(x) \rangle$, *kde* $C(x) = 1 + c_1 x + c_2 x^2 + ... + c_n x^n \in \mathbb{Z}_2[x]$ *sa nazýva ľavý charakteristický polynóm lineárnej diferenčnej rovnice. Ak stupeň ľavého charakteristického polynómu* $C(x)$ *je rovný* n *(t.j.* $c_n = 1$*), hovoríme, že register je nesingulárny. Hodnoty vnútorných stavov* $x_0, x_1, x_2, ..., x_{n-1}$ *nazývame počiatočné naplnenie registra.*

Fakt 1 *Nech* $C(x) \in \mathbb{Z}_2[x]$ *je ľavý charakteristický polynóm stupňa* n*. Ak je polynóm C(x) primitívny, potom každé z* $2^n - 1$ *nenulových naplnení príslušného LFSR generuje výstupnú postupnosť s periódou* $2^n - 1$.

Fakt 2 *Nech* $C(x) \in \mathbb{Z}_2[x]$ *je ľavý charakteristický polynóm stupňa* n*. Nech n-bitový vektor* $(x_{i-1}, x_{i-2}, ..., x_{i-n+1}, x_{i-n}) \in \mathbb{Z}_2^n$ *predstavuje vnútorné naplnenie príslušného registra. Potom sa zmena vnútorného stavu registra o 1 takt a nový bit* x_i *dá vyjadriť násobením matice typu* $n \times n$ *reprezentujúcej posun registra a vektora typu* $n \times 1$ *reprezentujúceho vnútorné naplnenie:*

$$\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 1 \\ c_n & c_{n-1} & c_{n-2} & \cdots & c_1 \end{pmatrix} \begin{pmatrix} x_{i-n} \\ x_{i-n+1} \\ \vdots \\ x_{i-2} \\ x_{i-1} \end{pmatrix} = \begin{pmatrix} x_{i-n+1} \\ x_{i-n+2} \\ \vdots \\ x_{i-1} \\ x_i \end{pmatrix}$$

LFSR sa v praxi používajú najmä pre ich jednoduchú hardvérovú implementáciu, dobré štatistické vlastnosti výstupných postupností a veľké periódy výstupných postupností. Avšak ich nevýhodou je, že sú lineárne - len zo znalosti $2n$ bitov výstupu je možné zostrojiť celý LFSR aj s počiatočným naplnením. Preto dochádza k ich spájaniu s nelineárnymi komponentami. Existuje viacero typov konštrukcií prúdových šifier založených na LFSR:

- Výstup jedného LFSR je filtrovaný nelineárnou funkciou.
- Taktovanie LFSR je riadené výstupom iného LFSR.
- Výstupy viacerých LFSR sú nelineárne kombinované do jedného výstupného bitu.
- Spätná väzba v registri je nelineárna.

V prácach [22], [28], [26] sú prezentované chybové útoky na základné typy prúdových šifier využívajúcich prvé 2 konštrukcie, t.j. nelineárne filtrovaný LFSR a LFSR s riadeným taktovaním. Samozrejme, existujú prúdové šifry, ktoré kombinujú uvedené konštrukčné metódy. Jednou z takýchto šifier je prúdová šifra LILI-128 [14]. Iné reálne prúdové šifry využívajú nelineárne spätnoväzobné posuvné registre (bod 4), napr. prúdová šifra Trivium [15].

2.2 Útoky postrannými kanálmi

Na kryptografické primitíva sa dá pozerať z dvoch uhlov pohľadu:

- Algoritmický pohľad - študuje sa samotný algoritmus, bezpečnosť príslušného primitíva závisí od bezpečnosti algoritmu - analýza bezpečnosti sa vykonáva štúdiom vlastností algoritmu.
- Implementačný pohľad - študuje sa konkrétna hardvérová/softvérová implementácia príslušného algoritmu. Analýza bezpečnosti sa vykonáva analýzou samotnej implementácie.

Je zrejmé, že to isté kryptografické primitívum môže mať viacero implementácií, či už sa jedná o šifrovacie/dešifrovacie zariadenie, generátor náhodných čísel, smart-kartu obsahujúcu kryptoprocesor a pamäť, atď. V prípade, že sa nájde chyba v algoritme, dá sa táto chyba zneužiť aj na útoky voči príslušným implementáciám. Avšak, ak sa nájde chyba/útok na určitú implementáciu, neznamená to, že tento útok bude uskutočniteľný aj na algoritmus vo všeobecnosti, resp. na iné implementácie.

V podstate teda existujú dva typy rôznych útokov na kryptografické primitívy:

- Priame útoky - útoky na algoritmus.
- Nepriame útoky - útoky na implementáciu algoritmu. Označujú sa aj ako *útoky postrannými kanálmi - angl. side-channel attacks.*

Nevýhodou nepriamych útokov je fakt, že sú špecifické len pre danú implementáciu; ich výhodou je, že sú vo všeobecnosti účinnejšie ako priame útoky - na vykonanie úspešného útoku vyžadujú menšie množstvo dát, či menší čas. Z tohto dôvodu sa dostávajú stále viac do centra pozornosti kryptoanalýzy, o čom svedčí množstvo publikácií na túto tému. Jednou z najznámejších je Quisquaterov článok: *Side-channel Attacks*([42]).

V tomto článku je prezentované základné delenie útokov postrannými kanálmi z hľadiska miery zásahu do zariadenia:

- Invazívne útoky - vyžadujú priamy zásah do komponentov hardvéru (šifrovacieho čipu, smart-karty, atď.), napríklad pripojenie sa na dátovú zbernicu s cieľom sledovania dátového toku. Keďže sledované časti zariadení (pamäť, procesor, zbernice) bývajú ukryté pod rôznymi vrstvami a špeciálnou pasívnou vrstvou, ktorá chráni čip pred sledovaním jeho správania, tieto útoky vyžadujú narušenie týchto vrstiev (angl. tzv. „depackaging“).
- Neinvazívne útoky - nevyžadujú priamy zásah do hardvéru, ale využívajú externe dostupné informácie o zariadení - napríklad jeho spotrebu, elektromagnetické vyžarovanie, dobu trvania výpočtov, atď.

- Polo-invazívne útoky - vyžadujú odstránenie ochranných vrstiev len po pasívnu vrstvu, na ich vykonanie nie je potrebný priamy vodivý kontakt s čipom.

Iné delenie z hľadiska činností útočníka:

- Aktívne útoky - aktívne ovplyvňujú správanie zariadenia a jeho funkčnosť, napríklad môže dochádzať k indukcii chýb počas výpočtov.
- Pasívne útoky - pasívne sledovanie správania zariadenia počas výpočtu bez zásahov a vyvolávania chýb.

Tieto delenia sa navzájom doplňujú, t.j. napríklad existujú invazívne pasívne útoky útoky - čítanie dát na zbernici, ku ktorej sa útočník dostane porušením pasívnej vrstvy - alebo existujú neinvazívne aktívne útoky - zmena vonkajšieho napájania zariadenia, ktorá priamo nezasahuje do zariadenia, avšak spôsobí chybu počas výpočtu.

Vo všeobecnosti platí, že neinvazívne útoky sú ľahšie realizovateľné pre svoje nižšie náklady v porovnaní s invazívnymi útokmi.

My sa v našej práci o.i. zaoberáme chybovou analýzou kryptografických primitív. Chybová analýza patrí do triedy aktívnych útokov na implementácie, t.j. aktívne sa do zariadenia/výpočtu vnesie chyba a následne sa zisťuje, aký vplyv má táto chyba na výpočet. Z príslušného chybného výpočtu sa potom snažíme zistiť tajné parametre kryptosystému.

2.2.1 Chybová analýza

Chybová analýza bola prvý krát použitá v roku 1996 kryptoanalytikmi Bonehom, Demillom a Liptonom [9] na útok voči kryptosystémom s verejným kľúčom založených na problémoch vyplývajúcich z teórie čísel (konkrétne išlo o útok na RSA, ktorý na uľahčenie modulárneho umocňovania používal čínsku zvyškovú vetu) a neskôr bola použitá Bihamom a Shamirom [7] ako základ útoku na súčinové blokové šifry (napríklad DES). Zatiaľ čo tieto techniky boli zovšeobecnené a aplikované na útoky voči iným blokovým šifrám a šifrovacím systémom s verejným kľúčom, donedávna existovalo málo výsledkov zameraných na podobné útoky na prúdové šifry. Jednou z prác zameraných na túto tému je článok [23], ktorý je výstupom diplomovej práce Jonathana Hocha [22] a diplomová práca autora tejto dizertačnej práce [26].

Chybové útoky, resp. indukcie fyzických chýb v implementáciách kryptosystémov, boli v minulosti úspešne vedené najmä v laboratórnych podmienkach, pričom väčšina z nich bola zameraná na blokové šifry a na kryptosystémy s verejným kľúčom. Rôzne techniky indukcie chýb v obvodoch, implementujúcich kryptosystém, môžu viesť na rôzne typy útokov.

Spôsobené chyby môžeme charakterizovať z viacerých hľadísk ([42]):

- Doba pôsobenia chyby
  1. Dočasné chyby - spôsobené chyby sú v zariadení prítomné počas výpočtu; po jeho skončení je možné zariadenie uviesť do pôvodného bezchybového stavu. Sem patria chyby, ktoré vzniknú pôsobením rádioaktivity, abnormálneho napájania, prípadne abnormálneho taktovania zariadenia. Ich výhodou je, že zariadenie je možné ďalej používať tak, ako pred útokom.
  2. Trvalé chyby - spôsobené chyby sa nedajú odstrániť, zariadenie sa tak bude chybne správať pri každom ďalšom použití. Sem patria chyby, ktoré vzniknú zmrazením zariadenia na konštantnú teplotu, či prerušením vodiča dátovej zbernice.
- Lokalita chyby - niektoré útoky vyžadujú presné umiestnenie chyby (napríklad konkrétne miesto v pamäti), iné útoky poskytujú väčšiu voľnosť pri umiestnení chyby (napríklad ľubovoľné miesto v pamäti).
- Moment indukcie chyby - niektoré chyby požadujú presný moment, v ktorom má prísť k indukcii (napríklad vzhľadom na stav výpočtu), iné útoky nie sú časovo obmedzené.
- Typ chyby - preklopenie bitu (bajtu), permanentná zmena bitu na 0 alebo 1, preklopenie v jednom smere, preskočenie inštrukcie, atď.

Najbežnejšie techniky indukcie chýb sú (prevzaté z [21]):

- Zmena napätia externého zdroja môže spôsobiť zlú interpretáciu inštrukcie alebo jej preskočenie. Táto metóda je často využívaná spoločnosťami vyvíjajúcimi smart-karty, no výskum sa deje za „zatvorenými dverami“, a teda existuje málo publikovaných výsledkov.
- Zmena externého taktovania obvodu môže spôsobiť vynechanie inštrukcie alebo prečítanie dát z inej časti pamäte.
- Zmena teploty obvodu (zvýšenie/zníženie nad/pod povolenú hranicu) môže spôsobiť náhodnú zmenu v pamäti RAM alebo zmenu možnosti čítať a zapisovať dáta (čítanie, resp. zápis dát môžu prestávať fungovať pri rôznych teplotách).
- Osvetlenie obvodu intenzívnym prúdom viditeľného svetla môže spôsobiť preklopenie niektorých bitov v registroch/pamäti.
- Osvetlenie obvodu laserovým lúčom má podobný efekt ako svetlo s tým rozdielom, že je ľahšia lokalizácia vzniknutej chyby. Navyše sa dá laser so zvýšenou intenzitou použiť aj na prepisovanie ROM pamäťových buniek alebo na zmenu logických hradiel.

- Použitie zväzku iónov alebo röntgenových lúčov môže taktiež viesť k indukcii chýb. Výhodou použitia tejto metódy je, že je neinvazívna, t.j. je ju možné použiť aj bez porušenia ochrannej vrstvy čipu/obvodu.
- Kozmické žiarenie taktiež dokáže spôsobiť preklopenie bitov (častý jav pri pamäťových zariadeniach vo vesmírnych staniciach).

Samozrejme, nie každé zariadenie je náchylné na všetky tieto útoky, resp. každý typ šifrovacieho hardvéru je citlivý voči inej technike. Napríklad fakt, že niektoré typy nevolatilných pamätí sú citlivé na nesymetrickú pravdepodobnosť preklopenia bitu (t.j. napríklad je väčšia pravdepodobnosť, že sa bit preklopí z jednotky na nulu ako naopak) za použitia elektromagnetického žiarenia, bol použitý Bihamom a Shamirom v útoku [7]. Naopak, volatilné SRAM pamäte majú symetrické rozdelenie pravdepodobnosti preklopenia bitu pri pôsobení ionizujúceho žiarenia. V práci [2] Anderson zase ukázal nízkonákladovú techniku nenáročnú na technické vybavenie, ktorá umožní útočníkovi získať fyzický prístup do kryptoprocesora (špeciálne ak ten je implementovaný na tzv. smart-karte), ktorý mu umožní indukovať chyby na konkrétnych miestach. Táto technika využíva stolový optický mikroskop, pomocou ktorého sa sústredí svetlo z blesku fotoaparáta na veľmi malú plochu integrovaného obvodu. Takto sa dajú ovplyvňovať dokonca jednotlivé bity vnútorného stavu kryptoprocesora. V poslednej dobe teda dochádza k prechodu od nákladných náročných chybových útokov ku lacnejším útokom, uskutočniteľným i v základných laboratórnych podmienkach.

Väčšina publikovaných útokov chybovou analýzou na šifrovacie zariadenia má charakter matematického modelu útoku, kde sa predpokladá, že útočník má k dispozícii šifrovacie/dešifrovacie zariadenie a príslušné laboratórne vybavenie pre indukciu chýb. Sleduje sa, aká je zložitosť daného typu útoku, t.j. koľko chýb je potrebné indukovať a aká je časová/pamäťová/dátová zložitosť útoku. Výsledky takéhoto výskumu tvoria cenný výstup pre dizajnérov hardvérových šifrátorov pri návrhu bezpečnej fyzickej implementácie.

Každý útok chybovou analýzou má tzv. **model útoku**. Ten popisuje, aké sú podmienky indukcie chýb, za ktorých je útok realizovateľný. Napríklad pri útoku pomocou preklopenia pamäťových bitov sa zväčša predpokladá, že útočník má len **čiastočnú** kontrolu nad lokalitou chyby, t.j. musí vypočítať, ktorý bit bol zmenený a taktiež, že je schopný vrátiť zariadenie do pôvodného bezchybového stavu, t.j. že chyby sú **dočasné** a dajú sa odstrániť reštartovaním zariadenia.

2.2.2 Ukážky chybových útokov

Prvým príkladom chybového útoku je útok na kryptosystém RSA, v ktorom je uľahčené modulárne umocňovanie pomocou Čínskej zvyškovej vety (ďalej len CRT z anglického Chinese Remainder Theorem), popísaný v práci [9].

Nech $N = pq$ je súčin dvoch veľkých prvočísel p, q. Nech $m \in \mathbb{Z}_N$ je správa a nech

$$C = m^d \bmod N$$

je podpis správy m, kde d je súkromný dešifrovací exponent (t.j. d je tajné). Kvôli efektívnosti väčšina implementácií RSA umocňuje m nasledovným spôsobom:
Najprv sa vypočítajú 2 časti:

$$d_p = d \bmod p \text{ a } d_q = d \bmod q,$$
$$C_p = m^{d_p} \bmod p \text{ a } C_q = m^{d_q} \bmod q.$$

Potom sa použije CRT na výpočet finálneho podpisu:

$$C = CRT(C_p, C_q) = m^d \bmod N.$$

Použitie CRT namiesto klasického umocňovania urýchli počítanie približne 4-krát.

Výsledná zašifrovaná správa je teda:

$$C = q(q^{-1} \bmod p)C_p + p(p^{-1} \bmod q)C_q \bmod N.$$

Predpokladajme teraz, že počas výpočtu C_p došlo k chybe a výsledkom umocňovania modulo p je $C_p^{'}$. Tým pádom je chybná aj výsledná správa $C^{'}$. Platí, že

$$C^{'} = C \bmod q \text{ a } C^{'} \neq C \bmod p.$$

Z toho vyplýva, že platí

$$GCD(C^{'} - C, N) = q,$$

čím sme vypočítali tajný parameter q, faktor verejného modulu N. Po zistení q navyše nie je problém dopočítať súkromný dešifrovací exponent d.

Druhým príkladom chybového útoku je chybový útok na dešifrovací exponent RSA, popísaný v práci [5].

Predpokladajme, že sme schopní preklopiť jeden náhodný bit súkromného kľúča d. Zašifrovaním správy m obdržíme správu c. Následne počas dešifrovania spôsobíme chybu, preklopenie jedného bitu súkromného kľúča a obdržíme správu $\hat{m}$. Ak predpokladáme, že sa bit $d[i]$ preklopil na bit $\overline{d[i]}$, potom:

$$\frac{\hat{m}}{m} = \frac{c^{2^i \overline{d[i]}}}{c^{2^i d[i]}} \bmod N,$$

a platí:

$$\frac{\hat{m}}{m} = \frac{1}{2^{c^i}} \bmod N \Rightarrow d[i] = 1 \text{ a } \frac{\hat{m}}{m} = 2^{c^i} \bmod N \Rightarrow d[i] = 0.$$

Tento proces zopakujeme, kým nezískame dostatočné množstvo bitov súkromného kľúča d. Navyše, tento útok sa dá použiť aj na kryptosystémy založené na probléme diskrétneho logaritmu.

Kapitola 3

Chybová analýza prúdových šifier

V tejto kapitole sú uvedené 2 články na tému chybovej analýzy prúdových šifier.

3.1 Útok fázovým posunom na prúdovú šifru Trivium

Prvý článok sa zaoberá chybovou analýzou šifry Trivium použitím techniky fázového posunu. Cieľom práce bolo zistiť, aká bude úspešnosť daného útoku a najmä, koľko bitov prúdového kľúča bude vyžadovať úspešný útok, t.j. aká bude dátová zložitosť. Výsledky sme porovnali s útokmi publikovanými v literatúre a zistili sme, že na úspešný útok fázovým posunom je potrebné vykonať 2 fázové posuny a vygenerovať prúdový kľúč o veľkosti 120 bitov, zatiaľ čo útok s najmenšou dátovou zložitosťou publikovaný v literatúre požadoval indukciu 2 jedno-bitových chýb a prúdový kľúč o veľkosti 420 bitov, čo je zhruba 70% zlepšenie.

3.1.1 Úvod

Chybová analýza je relatívne nový prístup v kryptoanalýze. Je založená na aktívnom zasahovaní do hardvérovej implementácie kryptosystému a vkladaní chýb do výpočtu, v dôsledku čoho sa výstup zariadenia líši od pôvodného bezchybného výstupu. Jednou z najznámejších prác na tému chybovej analýzy prúdových šifier je práca Hocha a Shamira [23]. V tejto práci autori navrhujú 2 spôsoby vkladania chýb do činnosti zariadení implementujúcich prúdové šifry - vkladanie jedno-bitových chýb do registrov zariadenia (t.j. preklápanie bitov v registroch) a fázový posun registrov. Preklápanie bitov spočíva v zmene hodnoty jedného alebo viacerých bitov v registroch zariadenia a fázový posun spočíva v manipulácii s taktovaním zariadenia, pričom sa vynúti posun jedného z registrov o 1 takt, zatiaľ čo ostatné registre stoja, čím dôjde k strate synchronizácie.

V roku 2008 skončil projekt eSTREAM zameraný na nájdenie prúdových šifier vhodných pre kryptografické aplikácie. Víťazné návrhy projektu boli rozdelené do 2

kategórií - prúdové šifry vhodné pre softvérovú implementáciu a prúdové šifry vhodné pre hardvérovú implementáciu. Jednou z vybraných šifier vhodných pre hardvérové použitie bola prúdová šifra Trivium [15].

Prvú publikovanú chybovú analýzou tejto prúdovej šifry vykonali Hojsík a Rudolf v roku 2008 [24]. Ich útok využíval vkladanie jedno-bitových chýb do vnútorného stavu šifry. Následne sa zostrojí sústava rovníc v premenných reprezentujúcich vnútorný stav zariadenia v danom čase. Vyriešením tejto sústavy sa nájde vnútorný stav šifry v danom čase, z ktorého nie je problém šifru taktovať spätne do inicializačného stavu, z ktorého sa priamo dá vyčítať tajný parameter - kľúč. Na tento útok autori potrebovali generovať prúdový kľúč o veľkosti 280 bitov a indukciu približne 43 jedno-bitových chýb. Neskôr svoj útok vylepšili použitím inej reprezentácie šifry, tzv. plávajúcej reprezentácie [25]. Výsledkom bolo zníženie počtu chýb potrebných pre nájdenie vnútorného stavu na približne 3.2 chyby a potrebovali na to generovať 800 bitov prúdového kľúča. Oba tieto útoky využívali len lineárne rovnice a na ich riešenie používali Gaussovu elimináciu.

Neskôr sa na kryptoanalýzu Trivia použila aj algebraická kryptoanalýza. Tá je založená na riešení systému nelineárnych rovníc viacerých premenných nad konečnými poliami reprezentujúcich kryptosystém [6], [11], [53]. Niektoré algoritmy algebraickej kryptoanalýzy boli už v minulosti použité pri analýze šifry Trivium, napr. [52], [36], [48]. V roku 2011 bola prvýkrát použitá aj pri chybovej analýze Trivia [39], kde sa na riešenie sústavy nelineárnych rovníc použili SAT-solvery. Vďaka tomuto vylepšeniu sa ďalej znížil potrebný počet indukovaných chýb na 2 chyby a počet bitov prúdového kľúča na 420.

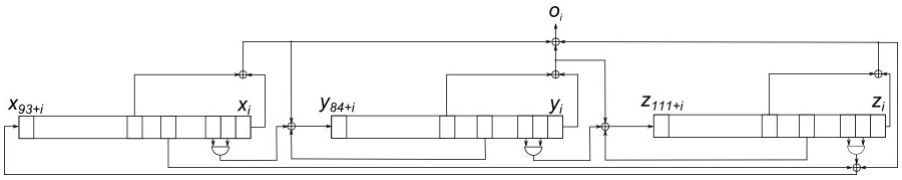
Nie je nám známe, že by bol doposiaľ publikovaný iný útok chybovou analýzou na šifru Trivium, než útok indukciou jedno-bitových chýb. Preto sme sa zamerali na útok fázovým posunom v kombinácii s rôznymi technikami algebraickej kryptoanalýzy: SAT-solverov, riešením sústavy pomocou Gröbnerových báz a algoritmu ElimLin.

Naše experimenty ukázali, že útok fázovým posunom dosahuje lepšie výsledky než útok preklopením bitov z hľadiska dátovej zložitosti. Zatiaľ čo najlepší útok pomocou preklopenia bitov požaduje indukciu 2 chýb a každý generovaný prúdový kľúč musí mať aspoň 420 bitov, náš útok požaduje fázový posun 2 registrov a každý generovaný prúdový kľúč musí mať aspoň 120 bitov. Tento výsledok len potvrdzuje fakt, že ochrana fyzickej implementácie šifrovacieho algoritmu je dôležitá a že ak má útočník možnosť ovplyvniť taktovanie zariadenia, resp. jeho komponentov, môže to viesť k útokom s nízkou dátovou a pamäťovou zložitosťou realizovateľných za pár sekúnd.

3.1.2 Chybová analýza šifry Trivium

Popis šifry Trivium

Trivium [15] je hardvérovo-orientovaná synchrónna prúdová šifra, ktorá generuje prúdový kľúč z 80-bitového tajného kľúča a 80-bitového inicializačného vektora. Jej vnútorný stav pozostáva z 288 bitov uložených v 3 nelineárnych spätnoväzobných posuvných registroch, zobrazených na obrázku 3.1.



Obr. 3.1: Plávajúca reprezentácia šifry Trivium v čase i

Vnútorný stav šifry IS_i v čase i:

$$IS_i = (x_{93+i}, ..., x_{1+i}, y_{84+i}, ..., y_{1+i}, z_{111+i}, ..., z_{1+i}), i \geq 0,$$

preto počiatočný stav šifry:

$$IS_0 = (x_{93}, ..., x_1, y_{84}, ..., y_1, z_{111}, ..., z_1).$$

Počiatočný stav je inicializovaný tajným kľúčom $K = (k_1, k_2, ..., k_{80})$ a inicializačným vektorom $IV = (u_1, u_2, ..., u_{80})$ nasledovne:

$$(\overbrace{k_1, k_2, ..., k_{80}, 0, 0, ..., 0}^{x}, \overbrace{u_1, u_2, ..., u_{80}, 0, 0, ..., 0}^{y}, \overbrace{0, 0, ..., 0, 1, 1, 1}^{z}).$$

Šifra sa po inicializácii registrov posunie o 1152 taktov bez toho, aby generovala výstup. Následne generuje 1 bit výstupu každý takt. Rovnice zodpovedajúce výstupu o_i a novým bitom registrov x_i, y_i, z_i sú:

$$o_i = x_i + x_{27+i} + y_i + y_{15+i} + z_i + z_{45+i}, \ i \geq 1, \tag{3.1}$$

$$\begin{aligned} x_{93+i} &= x_{24+i} + z_{45+i} + z_i + z_{1+i}z_{2+i}, \\ y_{84+i} &= y_{6+i} + x_{27+i} + x_i + x_{1+i}x_{2+i}, \\ z_{111+i} &= z_{24+i} + y_{15+i} + y_i + y_{1+i}y_{2+i}. \end{aligned} \tag{3.2}$$

Všetky rovnice sú nad konečným poľom $GF(2)$. Ak je známy vnútorný stav šifry v ľubovoľnom čase, nie je problém šifru taktovať v opačnom smere, t.j. spätne [39]. Ak teda poznáme vnútorný stav IS_{t_j} v čase t_j, sme schopní šifru vrátiť do počiatočného stavu a priamo z prvého registra zistiť tajný kľúč K.

Útok preklopením bitov

Ako bolo spomenuté v úvode, útok preklopením bitov je bežná metóda chybovej analýzy šifry Trivium. V podstate sa jedná o zmenu hodnoty jedného bitu niektorého z registrov alebo z 0 na 1, alebo z 1 na 0. Vo všeobecnosti sa dá takýto útok realizovať za nasledovných podmienok:

1. Útočník má k dispozícii šifrovacie zariadenie.
2. Útočník dokáže generovať prúdové kľúče ľubovoľnej dĺžky.
3. Útočník dokáže indukovať jedno-bitové chyby (t.j. preklopiť 1 bit v registri) a generovať príslušný chybový prúdový kľúč.
4. Útočník nemá kontrolu nad pozíciou chyby v registri.
5. Útočník má presnú kontrolu nad momentom indukcie chyby.
6. Útočník je schopný vrátiť zariadenie do počiatočného bezchybného stavu.
7. Útočník je schopný opakovať indukciu chyby neobmedzene.

Bod 4. znamená, že miesto indukcie chyby je útočníkovi neznáme, t.j. nepozná presnú pozíciu bitu, ktorý bol preklopený. Avšak táto pozícia sa dá presne určiť pomocou pôvodného bezchybného prúdového kľúča a príslušného chybového prúdového kľúča [24].

Bod 5. znamená, že útočník dokáže indukovať chyby vo zvolený časový moment, t.j. je schopný zariadenie zastaviť, indukovať chybu a zariadenie následne znovu uviesť do prevádzky. V kryptoanalýze šifry Trivium je cieľom nájsť vnútorný stav šifry v ľubovoľnom čase t_0, preto sa väčšinou chyby indukujú v momente t_0. Týmto momentom býva spravidla stav šifry po 1152 taktoch, keďže až od tohto bodu šifra generuje prúdový kľúč.

V tejto práci používame značenie prevzaté z práce [25]. Nech $\{x_i\}, \{y_i\}, \{z_i\}$ označujú postupnosti bitov prvého, druhého a tretieho registra. Nech $\{o_i\}$ označuje pôvodný bezchybný výstup (prúdový kľúč) Trivia a nech $\{o_i'\}$ označuje chybový prúdový kľúč vygenerovaný po vložení jedno-bitovej chyby. Označíme delta-postupnosti $\{\delta x_i\}, \{\delta y_i\}, \{\delta z_i\}$ a $\{\delta o_i\}$ ako diferencie medzi pôvodnými bezchybnými postupnosťami a chybovými postupnosťami. Všetky chybové premenné budú označené čiarkou. Použitím rovníc (3.1) a (3.2) a nasledujúceho vzťahu vyjadrujúceho diferenciu po násobení

$$\delta(ab) = a'b' + ab = \delta a \cdot b + a \cdot \delta b + \delta a \cdot \delta b$$

dostávame nasledovné rovnice nad $GF(2)$ pre delta-postupnosti:

$$\delta o_i = o_i + o_i' = \delta x_i + \delta x_{27+i} + \delta y_i + \delta y_{15+i} + \delta z_i + \delta z_{45+i},\ i \geq 1, \tag{3.3}$$

$$\begin{aligned}
\delta x_{93+i} &= \delta x_{24+i} + \delta z_{45+i} + \delta z_i + \delta z_{1+i} \cdot z_{2+i} + z_{1+i} \cdot \delta z_{2+i} + \delta z_{1+i} \cdot \delta z_{2+i}, \\
\delta y_{84+i} &= \delta y_{6+i} + \delta x_{27+i} + \delta x_i + \delta x_{1+i} \cdot x_{2+i} + x_{1+i} \cdot \delta x_{2+i} + \delta x_{1+i} \cdot \delta x_{2+i}, \\
\delta z_{111+i} &= \delta z_{24+i} + \delta y_{15+i} + \delta y_i + \delta y_{1+i} \cdot y_{2+i} + y_{1+i} \cdot \delta y_{2+i} + \delta y_{1+i} \cdot \delta y_{2+i}.
\end{aligned} \tag{3.4}$$

V momente t_0 pred indukciou chyby sú všetky delta-postupnosti registrových premenných rovné nule. Po indukcii chyby sa jedna z postupností zmení a bude obsahovať jednu jednotku podľa miesta indukcie. Napríklad, ak by došlo k chybe v prvom registri na pozícii j, potom $\delta x_j = 1$, zatiaľ čo $\delta x_i = 0$, pre $i \neq j$.

Následne vyjadríme $\{\delta x_i\}, \{\delta y_i\}, \{\delta z_i\}$ ako polynómy v premenných $\{x_i\}, \{y_i\}, \{z_i\}$ použitím rovníc (3.4). Tie sú dosadené do rovnice (3.3) a dostávame delta-rovnice pre prúdový kľúč [25]. Tieto rovnice spolu s (3.1) a (3.2) tvoria systém rovníc, ktorého vyriešením získame vnútorný stav IS_{t_j} v časovom momente t_j. Keďže používame plávajúcu reprezentáciu, v systéme máme premenné vyjadrujúce všetky vnútorné stavy od momentu t_0, t.j. teoreticky nemusíme hľadať stav t_0 ale niektorý nasledujúci stav.

V tejto reprezentácii sa po každom takte zvyšuje počet premenných. Na začiatku, v momente t_0, máme 288 premenných vyjadrujúcich stav IS_{t_0}. Po každom takte dostávame 3 nové premenné. Preto, po N taktoch, obsahuje systém rovníc $288 + 3N$ premenných.

Počet rovníc taktiež závisí od počtu taktov. Po každom takte dostávame jednu lineárnu rovnicu typu (3.1) a 3 nelineárne rovníce stupňa 2 typu (3.2), t.j. po N taktoch obsahuje systém $4N$ rovníc.

Počet a stupeň delta-rovníc pre prúdový kľúč závisí nielen od počtu taktov, ale aj od stupňa predchádzajúcich rovníc typu (3.4). Preto, ak indukujeme t chýb, tak počet rovníc označíme ako $\mathcal{O}(t)$.

Útok fázovým posunom

Fázový posun ako technika chybovej analýzy prúdových šifier bola navrhnutá v článku [23]. Spočíva vo vynútení straty synchronizácie registrov. Prúdové šifrátory väčšinou pozostávajú z viacerých registrov, ktoré sú alebo taktované súčasne, alebo výstup jedného registra ovplyvňuje taktovanie iného registra (angl. clock-controlled). Strata synchronizácie sa spôsobí manipuláciou s taktovaním, napr. sa jeden z registrov na jeden takt zastaví, alebo sa posunie o 1 takt pred ostatné registre. Táto technika bola použitá v niekoľkých útokoch na prúdové šifry, napr. [19], [34]. Vo všeobecnosti sa dá takýto útok realizovať za nasledovných podmienok:

1. Útočník má k dispozícii šifrovacie zariadenie.
2. Útočník dokáže generovať prúdové kľúče ľubovoľnej dĺžky.

3. Útočník dokáže manipulovať s taktovaním registrov, pričom alebo posunie jeden register o 1 takt pred ostatné registre, alebo ho naopak na 1 takt zastaví.
4. Útočník má presnú kontrolu nad momentom fázového posunu.
5. Útočník je schopný vrátiť zariadenie do počiatočného bezchybného stavu.
6. Útočník je schopný opakovať fázový posun neobmedzene s rôznymi registrami.

Bod 4. znamená, že útočník je schopný spustiť inicializáciu šifry Trivium. Následne zariadenie zastaví, posunie vybraný register o 1 takt a spustí generovanie (chybového) prúdového kľúča.

Na začiatku útoku, v momente t_0, tvoria vnútorný stav IS_{t_0} postupnosti $\{x_i\}_{i=1}^{93}, \{y_i\}_{i=1}^{84}, \{z_i\}_{i=1}^{111}$.

Predpokladajme, že si vynútime posun len prvého registra bez posunu ostatných 2 registrov o 1 takt. Novú premennú označíme $x_{94}^{(x)}$.

$$(x_{93}, x_{92}, ..., x_1) \rightarrow (x_{94}^{(x)}, x_{93}, ..., x_2),$$
$$x_{94}^{(x)} = x_{25} + z_{46} + z_1 + z_2 z_3.$$

Exponent $^{(x)}$ označuje premennú, ktorá vznikne po fázovom posune prvého registra. Ak by sme posunuli druhý register, premenná by bola označená exponentom $^{(y)}$ a ak by sme posunuli tretí register, bola by označená $^{(z)}$. Následne po prvom riadnom takte bude vnútorný stav šifry vyzerať nasledovne:

$$(x_{94}^{(x)}, x_{93}, ..., x_2, y_{84}, y_{83}, ..., y_1, z_{111}, z_{110}, ..., z_1)$$
$$\downarrow$$
$$(x_{95}^{(x)}, x_{94}^{(x)}, ..., x_3, y_{85}^{(x)}, y_{84}, ..., y_2, z_{112}^{(x)}, z_{111}, ..., z_2)$$

kde nové bity

$$x_{95}^{(x)} = x_{26} + z_{46} + z_1 + z_2 z_3,$$
$$y_{85}^{(x)} = y_7 + x_{29} + x_2 + x_3 x_4,$$
$$z_{112}^{(x)} = z_{25} + y_{16} + y_1 + y_2 y_3$$

a výstup

$$o_1^{(x)} = x_2 + x_{29} + y_1 + y_{16} + z_1 + z_{46}.$$

Vo všeobecnosti, ak dôjde k fázovému posunu prvého registra, potom rovnice majú tvar:

$$o_i^{(x)} = x_{1+i}^{(x)} + x_{28+i}^{(x)} + y_i^{(x)} + y_{15+i}^{(x)} + z_i^{(x)} + z_{45+i}^{(x)},\ i \geq 1, \tag{3.5}$$

$$\begin{aligned} x_{94+i}^{(x)} &= x_{25+i}^{(x)} + z_{45+i}^{(x)} + z_i^{(x)} + z_{1+i}^{(x)} z_{2+i}^{(x)}, \\ y_{84+i}^{(x)} &= y_{6+i}^{(x)} + x_{28+i}^{(x)} + x_{1+i}^{(x)} + x_{2+i}^{(x)} x_{3+i}^{(x)}, \\ z_{111+i}^{(x)} &= z_{24+i}^{(x)} + y_{15+i}^{(x)} + y_i^{(x)} + y_{1+i}^{(x)} y_{2+i}^{(x)}, \end{aligned} \tag{3.6}$$

$$x_{94}^{(x)} = x_{25} + z_{46} + z_1 + z_2 z_3. \tag{3.7}$$

Všetky rovnice sú nad $GF(2)$. Platí, že $\{x_i\}_{i=1}^{93} = \{x_i^{(x)}\}_{i=1}^{93}, \{y_i\}_{i=1}^{84} = \{y_i^{(x)}\}_{i=1}^{84}$, $\{z_i\}_{i=1}^{111} = \{z_i^{(x)}\}_{i=1}^{111}$. Rovnice pre fázový posun druhého alebo tretieho registra sú analogické k rovniciam (3.5), (3.6) a (3.7).

Na začiatku útoku, v čase t_0, je v systéme 288 premenných reprezentujúcich stav IS_{t_0}. Po fázovom posune dostaneme jedno novú premennú - nový bit posunutého registra. Ďalej, po každom riadnom takte dostávame 3 nové premenné. Preto, po N taktoch, systém rovníc obsahuje $3N + 288 + 1$ premenných.

Počet rovníc taktiež závisí od počtu taktov. Na začiatku, po fázovom posune, dostávame jednu rovnicu pre nový bit posunutého registra. Ďalej, po každom takte získame jednu lineárnu rovnicu typu (3.5) a 3 nelineárne rovnice druhého stupňa typu (3.6), t.j. po N taktoch získame $4N + 1$ rovníc.

3.1.3 Popis útoku fázovým posunom na Trivium

Cieľom útoku je nájsť vnútorný stav IS_{t_0} šifry Trivium v danom čase t_0 použitím fázového posunu. Na začiatku vygenerujeme N bitov prúdového kľúča a do systému rovníc vložíme príslušné rovnice (3.1), (3.2). Označme tento systém rovníc ako S. Následne zariadenie vrátime do stavu IS_{t_0}, aplikujeme fázový posun jedného z 3 registrov a znovu vygenerujeme N bitov prúdového kľúča a pridáme príslušné rovnice (3.5), (3.6), (3.7) do systému rovníc S. Následne znovu vrátime zariadenie do stavu IS_{t_0}, aplikujeme fázový posun iného z 3 registrov a znovu príslušné rovnice vložíme do systému S. Počet rôznych fázových posunov označujeme ako m. Algoritmus útoku je označený ako algoritmus 1.

Na riešenie systému rovníc S sme sa rozhodli použiť metódy algebraickej kryptoanalýzy. Vybrali sme si 2 známe metódy: SAT-solvery a Gröbnerove bázy a relatívne nový algoritmus ElimLin [10]. SAT-solvery boli už použité pri chybovej analýze šifry Trivium v práci [39] v spojení s preklápaním bitov. Taktiež sa využívajú pri kryptoanalýze iných systémov, napr. Keeloq [12] a Bivium [16], čo je variant šifry Trivium. Algoritmus ElimLin je jednoduchý algoritmus na riešenie systému rovníc o viacerých premenných nad konečnými poliami. Keďže sa jedná o nový algoritmus, zvolili sme pre jeho porovnanie SAT-solvery. V prípade, že sme pomocou tejto metódy neboli schopní nájsť riešenie sústavy, ale len zredukovať počet rovníc, zvyšné rovnice sme sa pokúsili vyriešiť pomocou Gröbnerových báz. Algoritmus riešenia systému rovníc sme označili SOLVE a je predstavený ako algoritmus 2.

Algoritmus 1: Útok fázovým posunom na Trivium

Vstup: Počet fázových posunov m, počet bitov prúdového kľúča N.
Výstup: Kľúč K alebo prázdna množina, ak K nebol nájdený.

1: Nastav systém rovníc $S \leftarrow \emptyset$
2: Dostaň zariadenie do stavu IS_{t_0}
3: Generuj N prúdového kľúča zo stavu IS_{t_0}
4: Vlož rovnice (3.1) a (3.2) do systému S.
5: **for** i = 1 **to** m **do**
6: Vráť zariadenie do stavu IS_{t_0}.
7: Urob fázový posun predtým neposúvaného registra o 1 takt.
8: Generuj N bitov prúdového kľúča.
9: Vlož rovnice (3.5), (3.6) a (3.7) do systému S.
10: **end for**
11: Riešenie $\leftarrow$ SOLVE(S)
12: **if** Riešenie $\neq \emptyset$ **then**
13: Taktuj zariadenie zo stavu IS_{t_0} späť do počiatočného stavu.
14: Čítaj kľúč K z prvého registra.
15: **return** K.
16: **else**
17: **return** $\emptyset$.
18: **end if**

Algoritmus 2 rieši systém rovníc. Ako bolo spomínané, používame 2 metódy riešenia - SAT-solvery a algoritmus ElimLin v spojení s Gröbnerovými bázami. Vstupom je systém rovníc nad konečným poľom $GF(2)$. Vstupom SAT-solverov je systém rovníc konvertovaný na problém splniteľnosti konjunktívnej normálnej formy a výstupom je priradenie logických hodnôt premenným. Vstupom algoritmu pre hľadanie Gröbnerových báz sú rovnice v algebraickej normálnej forme, výstupom je riešenie rovníc. Vstupom algoritmu ElimLin je sústava rovníc, pričom algoritmus iteratívne vykonáva 2 fázy. V prvej fáze nájde všetky lineárne rovnice v lineárnom obale pôvodných rovníc. V druhej fáze vyjadrí z týchto lineárnych rovníc premenné a dosadí ich do celého systému, kým sa nevyčerpajú všetky lineárne rovnice. Tento proces sa iteratívne opakuje, pokiaľ v lineárnom obale už nezostane ani jedna lineárna rovnica. Následne algoritmus vráti 2 systémy rovníc: systém lineárnych rovníc (v našom algoritme 2 označený L) a upravený pôvodný systém rovníc (v algoritme 2 označený S) [10].

Taktiež sme nakoniec skombinovali útok fázovým posunom a preklopením bitov v snahe ďalej znížiť počet bitov prúdového kľúča potrebný pre úspešné vykonanie útoku. Tento upravený algoritmus sme označili ako algoritmus 3.

Algoritmus 2: SOLVE(S) - Algoritmus riešenia systému rovníc S.

Vstup: Systém rovníc S.
Výstup: Hodnota IS_{t_0} alebo prázdna množina, ak IS_{t_0} nebol nájdený.

```
1: Riešenie ← ∅.
2: L ← ∅.
3: if Metóda riešenia je SAT-solver then
4:   Riešenie ← Rieš S pomocou SAT-solvera.
5:   if Nenájdené riešenie then
6:     return ∅.
7:   end if
8:   return Riešenie.
9: end if
10: if Metóda riešenia je ElimLin then
11:   L, S ← ElimLin(S)
12:   if S = ∅ then
13:     L ← Rieš L Gaussovou eliminániou.
14:     if Nenájdené riešenie L then
15:       return ∅.
16:     end if
17:     Vráť L.
18:   else
19:     S ← Gröbner(S).
20:     L ← L ∪ S.
21:     L ← Rieš L Gaussovou eliminániou.
22:     if Nenájdené riešenie L then
23:       return ∅.
24:     end if
25:     return L.
26:   end if
27: end if
```

Algoritmus 3: Útok fázovým posunom na Trivium s preklopením bitov

Vstup: Počet fázových posunov m, počet bitov prúdového kľúča N, počet preklopených bitov t.

Výstup: Kľúč K alebo prázdna množina, ak K nebol nájdený.

1: Nastav systém rovníc $S \leftarrow \emptyset$
2: Dostaň zariadenie do stavu IS_{t_0}
3: Generuj N bitov prúdového kľúča zo stavu IS_{t_0}
4: Vlož rovnice (3.1) a (3.2) do systému S.
5: **for** i = 1 **to** m **do**
6: Vráť zariadenie do stavu IS_{t_0}.
7: Urob fázový posun predtým neposúvaného registra o 1 takt.
8: Generuj N bitov prúdového kľúča.
9: Vlož rovnice (3.5), (3.6) a (3.7) do systému S.
10: **end for**
11: **for** i=1 **to** t **do**
12: Vráť zariadenie do stavu IS_{t_0}.
13: Preklop náhodný bit v stave IS_{t_0}.
14: Generuj N chybového prúdového kľúča.
15: $e \leftarrow$ Odhadni pozíciu chyby.
16: Vypočítaj delta-rovnice (3.3) pre e a vlož ich do systému S.
17: **end for**
18: Riešenie $\leftarrow$ SOLVE(S)
19: **if** Riešenie $\neq \emptyset$ **then**
20: Taktuj zariadenie zo stavu IS_{t_0} späť do počiatočného stavu.
21: Čítaj kľúč K z prvého registra.
22: **return** K.
23: **else**
24: **return** $\emptyset$.
25: **end if**

3.1.4 Výsledky experimentov

Na implementáciu experimentov sme použili zdarma dostupný open-source matematický softvérový systém SAGE[1]. Tento systém podporuje prácu s Booleovskými polynómami a priamo implementuje použité algoritmy - SAT-solvery, algoritmus ElimLin a algoritmus na hľadanie Gröbnerových báz. Všetky experimenty boli vykonané na štandardnom PC (OS Kubuntu 12.04, CPU Intel Core 2 Duo E8400 @ 3.00 GHz, 4 GB RAM).

Pri každom útoku nás zaujímala jeho *dátová zložitosť*, t.j. koľko bitov prúdového kľúča potrebujeme na úspešné vykonanie útoku. Taktiež nás zaujímala pamäťová zložitosť, ktorú sme vyjadrili v počte rovníc systému S. Keďže nás zaujímali len útoky, pri ktorých riešenie sústavy trvalo maximálne pár minút, každý útok, pri ktorom riešenie sústavy trvalo dlhšie sme prerušili a vyhlásili za neúspešný.

Fázový posun jedného registra

Pre prvý útok sme použili algoritmus 1 s počtom fázových posunov $m = 1$ a sledovali sme počet bitov prúdového kľúča N potrebný na úspešné nájdenie vnútorného stavu IS_{t_0}.

Vykonali sme 3 varianty útoku - v každej sme vykonali fázový posun iného registra. Pre každý register sme útok 100 krát zopakovali pri použití SAT-solverov a 100 krát pri použití algoritmu ElimLin. Sledovali sme jeho úspešnosť - pomer medzi počtom úspešných nájdení stavu IS_{t_0} a medzi celkovým počtom útokov.

Výsledky uvádzame v tabuľke 3.1. Stĺpec „1.“ reprezentuje fázový posun prvého registra, stĺpec „2.“ reprezentuje fázový posun druhého registra a stĺpec „3.“ reprezentuje fázový posun tretieho registra.

	1.		2.		3.	
N	SAT	ElimLin	SAT	ElimLin	SAT	ElimLin
100 bitov	0 / 100	0 / 100	0 / 100	0 / 100	0 / 100	0 / 100
200 bitov	0 / 100	0 / 100	0 / 100	0 / 100	0 / 100	0 / 100
300 bitov	0 / 100	0 / 100	0 / 100	0 / 100	0 / 100	0 / 100

Tabuľka 3.1: Fázový posun jedného registra

Pamäťová zložitosť: $M = 8N + 1$ rovníc o $289 + 6N$ premenných:

- Rovnice (3.1),(3.2): $N + 3N$ v $288 + 3N$ premenných.
- Rovnice (3.5),(3.6),(3.7): $N + 3N + 1$ v $3N + 1$ premenných.

[1]Dostupný na http://www.sagemath.org

Dátová zložitosť: $D = 2N$ bitov prúdového kľúča.

Ako vidno z tabuľky 3.1, žiadny útok nebol úspešný pri počtoch bitov 100, 200 a 300. Vo všetkých prípadoch trvalo riešenie príslušnej sústavy viac ako pár minút a útok bol zastavený. Preto sme sa pokúsili útok vylepšiť použitím fázového posunu 2 registrov.

Fázový posun dvoch registrov

Ako druhý útok sme použili algoritmus 1 s počtom fázových posunov $m = 2$ a sledovali sme počet bitov prúdového kľúča N potrebný na úspešné nájdenie vnútorného stavu IS_{t_0}.

Vykonali sme 3 varianty útoku - v každej sme vykonali fázový posun inej dvojice registrov. Vo všetkých prípadoch sme vykonali 100 útokov pomocou SAT-solverov a 100 útokov pomocou algoritmu ElimLin. Sledovali sme jeho úspešnosť - pomer medzi počtom úspešných nájdení stavu IS_{t_0} a medzi celkovým počtom útokov.

Výsledky uvádzame v tabuľke 3.2. Stĺpec „1. & 2.“ reprezentuje fázový posun prvého registra a druhého registra, atď.

	1. & 2.		2. & 3.		1. & 3.	
N	SAT	ElimLin	SAT	ElimLin	SAT	ElimLin
120 bitov	100 / 100	0 / 100	100 / 100	0 / 100	100 / 100	0 / 100
130 bitov	100 / 100	100* / 100	100 / 100	100* / 100	100 / 100	100* / 100
140 bitov	100 / 100	100* / 100	100 / 100	100* / 100	100 / 100	100* / 100
150 bitov	100 / 100	100 / 100	100 / 100	100 / 100	100 / 100	100 / 100

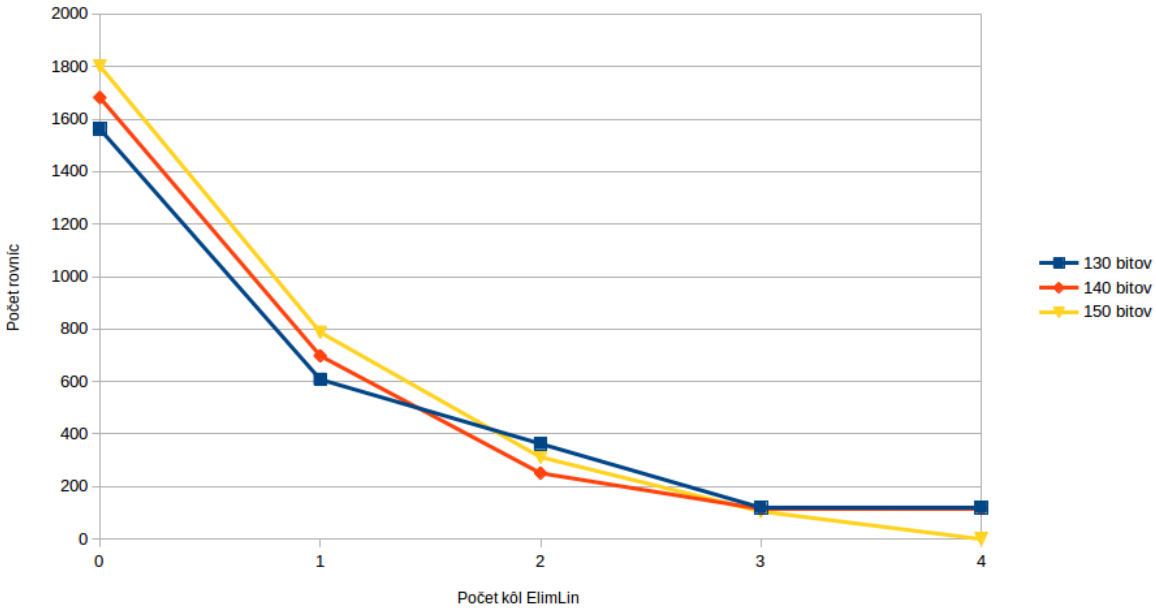
Tabuľka 3.2: Fázový posun dvoch registrov

* Vo všetkých prípadoch útoku so 130 a 140 bitmi prúdového kľúča algoritmus ElimLin zredukoval počet rovníc, avšak nevyriešil systém S. Preto sme na redukovaný systém rovníc aplikovali algoritmus pre hľadanie Gröbnerových báz pre nájdenie riešenia.

Ako možno vidieť v tabuľke, tento útok bol úspešný už so 120 bitmi prúdového kľúča za použitia SAT-solverov, so 130 bitmi za použitia algoritmu ElimLin a následného dohľadania riešení pomocou Gröbnerových báz a so 150 bitmi za použitia len algoritmu ElimLin.

Pamäťová zložitosť: $M = 12N + 2$ rovníc o $290 + 9N$ premenných:

- Rovnice (3.1),(3.2): $N + 3N$ o $288 + 3N$ premenných.
- Rovnice (3.5),(3.6),(3.7): $2(N + 3N + 1)$ o $2(3N + 1)$ premenných.



Obr. 3.2: ElimLin: Počet rovníc po každom kole (posun 2 registrov)

Dátová zložitosť: $D = 3N$ bitov prúdového kľúča.

Redukcia počtu rovníc v algoritme ElimLin po každej jeho iterácii je zobrazená v grafe na obrázku 3.2. Vo všetkých prípadoch nebol ElimLin schopný po 5 kolách extrahovať ďalšie lineárne rovnice. V prípade $N = 150$ vyriešil systém S úplne, v prípade $N = 130$ a $N = 140$ zostalo po 5 kolách menej ako 200 rovníc, ktoré sme vyriešili pomocou algoritmu pre hľadanie Gröbnerových báz.

Útok fázovým posunom a preklopením bitov

Ako tretí útok sme použili algoritmus 3 s počtom fázových posunov $m = 1$ a počtom indukovaných jedno-bitových chýb t. Znovu sme vytvorili 3 varianty, podľa toho, na ktorý register bol fázový posun aplikovaný. Útok fázovým posunom s posunom 1 registra nebol úspešný pre žiadnu veľkosť prúdového kľúča N. Preto sme sa pokúsili zlepšiť úspešnosť pridaním chýb pomocou preklopenia bitov. Uskutočnili sme niekoľko simulácií pre rôzne hodnoty t a pozorovali sme, aká je minimálna hodnota N potrebná na 100% úspešnosť útoku. Výsledky sú zhrnuté v tabuľke 3.3.

Ako je vidno v tabuľke, útok fázovým posunom jedného registra potrebuje 2 ďalšie preklopené bity, aby bol úspešný - avšak len pri použití SAT-solverov. Najmenší počet bitov N potrebný pre zlomenie kryptosystému bol experimentálne určený na $N = 120$ za použitia $t = 200$ chýb.

	1.		2.		3.	
t	SAT	ElimLin	SAT	ElimLin	SAT	ElimLin
2 chyby	460	-	450	-	470	-
3 chyby	420	450	420	450	420	450
4 chyby	380	400	380	400	380	400
10 chýb	320	340	320	340	320	340
20 chýb	200	220	200	220	200	220
50 chýb	170	180	170	180	170	180
100 chýb	140	150	140	150	140	150

Tabuľka 3.3: Útok fázovým posunom a preklopením bitu

Je dobré spomenúť, že algoritmus ElimLin vo všetkých prípadoch (s výnimkou 2 indukovaných chýb) postačoval na vyriešenie sústavy S pre príslušné voľby t a N v tabuľke.

Pamäťová zložitosť: $M = 8N + 1 + \mathcal{O}(t)$ rovníc o $289 + 6N$ premenných:

- Rovnice (3.1),(3.2): $N + 3N$ o $288 + 3N$ premenných.
- Rovnice (3.3): $\mathcal{O}(t)$.
- Rovnice (3.5),(3.6),(3.7): $N + 3N + 1$ o $3N + 1$ premenných.

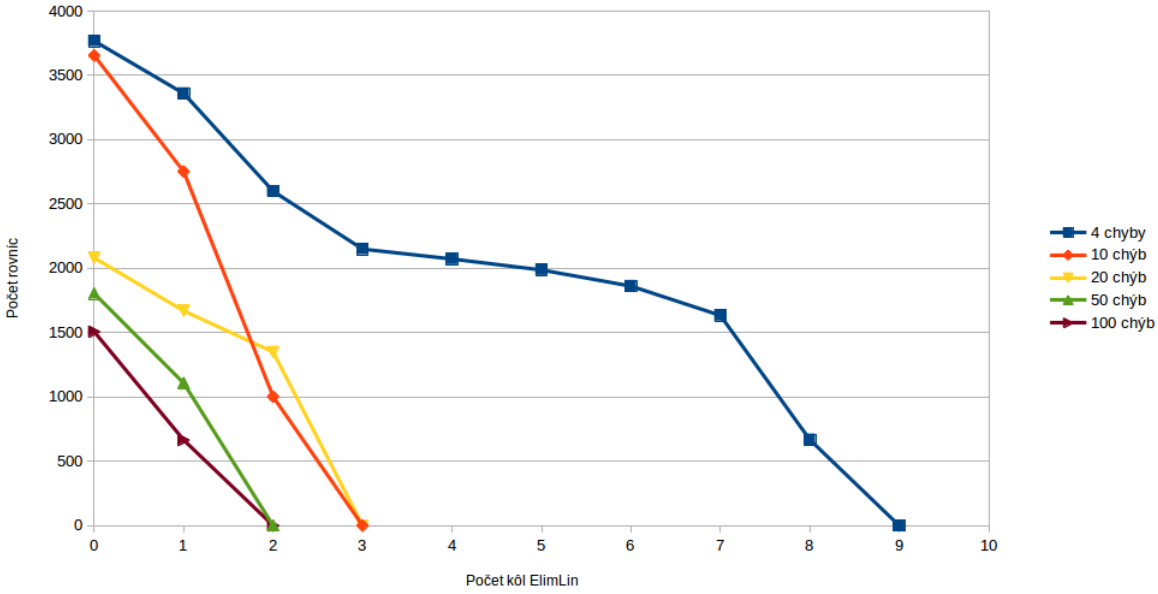
Dátová zložitosť: $D = (2 + t)N$ bitov prúdového kľúča.

Redukcia počtu rovníc v algoritme ElimLin po každej jeho iterácii je zobrazená v grafe na obrázku 3.3. Počet rovníc v grafe je priemernou hodnotou pre každý prípad. Z grafu je vidieť, že algoritmus našiel riešenie vo všetkých prípadoch (dotýka sa osi x). Taktiež je vidieť, že počet iterácii algoritmu ElimLin je nepriamo úmerný počtu preklopených bitov.

3.1.5 Záver

Uviedli sme výsledky našej simulácie chybovej analýzy prúdovej šifry Trivium pomocou fázového posunu v kombinácii s algebraickou kryptoanalýzou. Naše experimenty ukázali, že fázový posun 2 registrov a prúdové kľúče o veľkosti 120 bitov postačujú na úspešné vykonanie útoku. Doteraz najlepší útok chybovou analýzou na šifru Trivium požadoval indukciu 2 jedno-bitových chýb a prúdové kľúče o veľkosti 420 bitov.

Taktiež sme skúmali kombináciu fázového posunu a preklopenia bitov. Zistili sme, že na nájdenie vnútorného stavu šifry s prúdovými kľúčmi o veľkosti 120 bitov potrebujeme indukovať až 200 jedno-bitových chýb a vykonať fázový posun niektorého z registrov o 1 takt.



Obr. 3.3: ElimLin: Počet rovníc po každom kole

Naše výsledky ukazujú, že nezáleží na tom, na ktorý z registrov aplikujeme fázový posun - výsledky boli vo všetkých prípadoch prakticky rovnaké. Taktiež, výsledky za použitia SAT-solverov sú málo odlišné od výsledkov za použitia algoritmu ElimLin, čo je pre tento relatívne nový algoritmus sľubný výsledok.

Výhodou útoku fázovým posunom nad preklopením bitov je fakt, že zatiaľ čo pri preklopení bitu musí útočník zistovať, na ktorom mieste k preklopeniu prišlo, pri fázovom posune to nie je potrebné. Samozrejme, samotná fyzická implementácia tohto typu útoku môže byť problém a zatiaľ žiadna implementácia nebola predstavená v dostupnej literatúre. V práci [19] autori tvrdia, že takýto útok by sa dal vykonať na zariadenie pracujúce v testovacom móde, v ktorom je používateľ zväčša schopný pracovať s jednotlivými komponentami zariadenia samostatne, t.j. by mohol posúvať jednotlivé registre o ľubovoľný počet taktov. Taktiež autori tvrdia, že takýto útok by sa dal implementovať ako „zadné dvierka“ do zariadenia jeho výrobcom.

Samozrejme, ak je taktovanie a jednotlivé prepojenia častí zariadenia bezpečne implementované a chránené a zariadenia sú dizajnované so zreteľom na bezpečnosť, takýto chybový útok nie je reálnou hrozbou. Avšak, ak je útočník schopný manipulovať s taktovaním zariadenia, môže vykonať úspešný útok s nízkou pamäťovou a dátovou zložitosťou, ktorého finálna fáza počítania vnútorného stavu trvá len pár sekúnd.

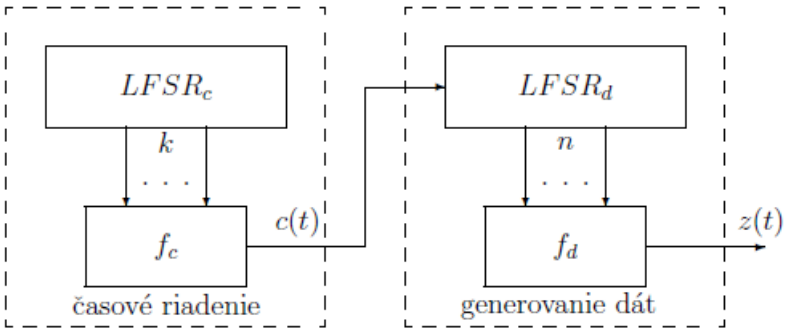
3.2 Chybová analýza prúdovej šifry LILI-128

Druhý článok sa zaoberá chybovou analýzou prúdovej šifry LILI-128. Vychádzal z článku [23], v ktorom autori prezentovali okrem techník chybovej analýzy aj útok na túto šifru pomocou metódy preklápania bitov. Naším cieľom bolo zostrojiť útok fázovým posunom a zistiť, či to pomôže znížiť počet indukovaných chýb potrebný na úspešné vykonanie útoku. Výsledkom bolo 20-násobné zníženie počtu indukovaných chýb.

3.2.1 Prúdová šifra LILI-128

Prúdová šifra LILI-128 [14] bola jednou zo šiestich prúdových šifier predložených do európskeho výskumného kryptografického projektu NESSIE. Je to prúdová šifra s nelineárne filtrovaným dátovým registrom, ktorého taktovanie je riadené výstupom časového registra. Jedná sa teda o kombináciu dvoch typov konštrukcie prúdových šifier založených na lineárnych spätnoväzobných registroch, ktoré boli predstavené v kapitole 2.1.

Prúdová šifra LILI-128 pozostáva z dvoch lineárnych spätnoväzobných registrov: 39-bitového časového registra $LFSR_C$ a 89-bitového dátového registra $LFSR_D$. Kľúč tejto šifry má 128 bitov a používa sa na inicializáciu registrov. Prvých 39 bitov kľúča sa použije ako počiatočné naplnenie časového registra a zvyšných 89 bitov kľúča sa použije ako počiatočné naplnenie dátového registra. V prípade, že by malo byť počiatočné naplnenie časového alebo dátového registra nulové (postupnosť nulových bitov) toto naplnenie je považované za neplatné.



Obr. 3.4: Prúdová šifra LILI-128 (prevzané z [14])

Na obrázku 3.4 je vidieť schému šifry LILI-128 (resp. generátora prúdového kľúča LILI-128). Dá sa rozdeliť na dva podsystémy: časové riadenie a generovanie dát, resp. prúdového kľúča. Lineárny spätnoväzobný register pre podsystém časového riadenia je taktovaný pravidelne. Výstupom tejto časti je postupnosť celých čísel, ktoré ovládajú

taktovanie lineárneho spätnoväzobného registra v podsystéme generovania dát.

Samotné generovanie prúdového kľúča prebieha nasledovne:

- Na 10 bitov dátového registra $LFSR_D$ sa aplikuje nelineárna filtrovacia funkcia f_d, ktorej výstupom je jeden bit tvoriaci súčasť prúdového kľúča.
- Časový register $LFSR_C$ sa posunie o jeden takt. Podľa jeho dvoch bitov sa určí celé číslo c z množiny $\{1, 2, 3, 4\}$.
- Dátový register sa posunie o c taktov.

Podsystém časového riadenia

Podsystém časového riadenia LILI-128 využíva pseudonáhodnú binárnu postupnosť vyprodukovanú pravidelne taktovaným registrom $LFSR_C$ dĺžky 39 a funkciu f_c, ktorej vstupy tvoria 2 bity $LFSR_C$ ($k = 2$) a ktorej výstupom je celé číslo z množiny $\{1, 2, 3, 4\}$. $LFSR_C$ má asociovaný primitívny charakteristický polynóm:

$$x^{39} + x^{35} + x^{33} + x^{31} + x^{17} + x^{15} + x^{14} + x^2 + 1.$$

Ako bolo spomínané, nulové počiatočné naplnenie registra je považované za neplatné. $LFSR_C$ produkuje binárnu postupnosť s periódou $P_C = 2^{39} - 1$.

Nech sú počiatočné bity registra $LFSR_C$ označené ako $s[38], s[37], ..., s[1], s[0]$ (zľava doprava) a nech sa register posúva vpravo. Potom, v čase t (v prvom okamihu platí $t = 0$) sa hodnota bitu na poslednej pozícií (najviac vľavo) vypočíta:

$$s[39+t] = s[37+t] \oplus s[25+t] \oplus s[24+t] \oplus s[22+t] \oplus s[8+t] \oplus s[6+t] \oplus s[4+t] \oplus s[t]$$

Nech sú jednotlivé bity registra $LFSR_C$ (v ľubovoľnom čase) označené ako $x_{38}, x_{37}, ..., x_1, x_0$. Vstup do funkcie f_c v čase t tvoria bity x_{12} a x_{20} registra (t.j. bity $s[12+t]$ a $s[20+t]$) a výstupom funkcie v čase t je celé číslo $c(t)$, pričom $c(t) \in \{1, 2, 3, 4\}$. Predpis funkcie $f_c : \mathbb{Z}_2^2 \to \{1, 2, 3, 4\}$:

$$f_c(x_{12}, x_{20}) = 2(x_{12}) + x_{20} + 1$$

Táto funkcia je bijektívne zobrazenie a rozdelenie celých čísel $c(t)$ je blízke rovnomernému rozdeleniu za predpokladu, že rozdelenia bitov x_{12} a x_{20} sú nezávislé a rovnomerné.

Podsystém generovania dát

Podsystém generovania dát využíva postupnosť celých čísel $c(t)$, ktorú produkuje podsystém časového riadenia na ovládanie taktovania dátového registra $LFSR_D$ dĺžky 89. Obsah 10 bitov registra $LFSR_D$ potom tvorí vstup do funkcie f_d, ktorá je daná tabuľkou. Výstupom funkcie f_d je bit prúdového kľúča $z(t)$. Nasleduje posunutie oboch registrov o príslušný počet taktov a celý proces sa zopakuje na vyprodukovanie ďalšieho bitu prúdového kľúča $z = \{z(t)\}_{t=1}^{\infty}$.

Dátový register $LFSR_D$ má asociovaný primitívny charakteristický polynóm:

$$x^{89} + x^{83} + x^{80} + x^{55} + x^{53} + x^{42} + x^{39} + x + 1.$$

Znovu je dobré zdôrazniť, nulové počiatočné naplnenie registra je považované za neplatné. Perióda binárnej postupnosti vygenerovanej dátovým registrom je $P_D = 2^{89} - 1$.

Nech sú počiatočné pozície registra $LFSR_D$ označené ako $u[88], u[87], ..., u[1], u[0]$ (zľava doprava) a nech sa register posúva vpravo. Potom, v čase t (v prvom okamihu platí $t = 0$) sa hodnota bitu na poslednej pozícií (najviac vľavo) vypočíta:

$$u[89+t] = u[88+t] \oplus u[50+t] \oplus u[47+t] \oplus u[36+t] \oplus u[34+t] \oplus u[9+t] \oplus u[6+t] \oplus u[t].$$

Nech sú jednotlivé bity registra $LFSR_D$ (v ľubovoľnom čase) označené ako $x_{88}, x_{87}, ..., x_1, x_0$. Do funkcie f_d vstupujú bity registra $x_0, x_1, x_3, x_7, x_{12}, x_{20}, x_{30}, x_{44}, x_{65}, x_{80}$. Funkcia f_d bola autormi skonštruovaná podľa techniky prezentovanej v [46].

3.2.2 Pôvodný chybový útok na LILI-128

V práci [23] autori prezentovali chybový útok na šifru LILI-128. V podstate sa jedná o útok indukovaním chýb s nízkou Hammingovou váhou v dátovom registri. Cieľom útoku je zistiť počiatočné naplnenia oboch registrov.

Model útoku predpokladá nasledovné:

- Útočník má k dispozícii šifrovacie zariadenie a je schopný generovať prúdový kľúč.
- Útočník je schopný vkladať jedno-bitové chyby, pričom nemá kontrolu nad ich lokalitou.
- Útočník je schopný zariadenie reštartovať do pôvodného stavu.

Prvou fázou útoku je indukcia jedno-bitových chýb na náhodných miestach v dátovom registri a vygenerovanie príslušného chybového prúdového kľúča. Následne sa zariadenie uvedie do stavu pred indukciou chýb a generovaním výstupu (t.j. „zresetuje sa“) a postup sa zopakuje, kým nezískame 89 rôznych prúdových kľúčov, čo zodpovedá indukcii jedno-bitovej chyby v každom bite registra. S veľkou pravdepodobnosťou sa tento proces bude opakovať viac ako 89-krát, keďže sa môže stať, že v 2 rôznych pokusoch dôjde k indukcii chyby na tom istom mieste. Celý tento postup generácie 89 rôznych chybových prúdových kľúčov následne opakujeme, avšak pred indukciou chyby posunieme celú konštrukciu o jeden takt. Pozorujeme, že množiny prúdových kľúčov obsahujú niekoľko rovnakých prúdových kľúčov. Je to spôsobené tým, že v prípade, že indukovaná chyba nebola indukovaná na mieste, ktoré ovplyvňuje nový bit (podľa diferenčnej rovnice), je jedno, či chybu indukujeme na mieste i, a potom

zariadenie posunieme o jeden takt, alebo ho najprv posunieme o jeden takt a následne indukujeme chybu na pozícii $i-1$. Spočítaním, koľko prúdových kľúčov sa v daných množinách zhoduje, vieme zistiť, či bol register $LFSR_D$ posunutý o 1, 2, 3 alebo 4 takty, čím získame 2 bity pôvodného naplnenia registra $LFSR_C$, resp. získame 2 lineárne rovnice nad GF(2) nad pôvodným naplnením registra $LFSR_C$. Čiže po zhruba 20 opakovaniach (indukcií chýb po zhruba 20 rôznych posunoch šifry) sme schopní vypočítať pôvodné počiatočné naplnenie časového registra.

Po určení počiatočného naplnenia časového registra použijeme algoritmus [23] na nájdenie počiatočného naplnenia dátového registra, pričom použijeme už vygenerované chybové prúdové kľúče. Samotný útok sa dá algoritmicky zapísať nasledovne:

Algoritmus 4: Chybový útok na LILI-128 (Hoch, Shamir)

1: Vygeneruj bezchybový prúdový kľúč
2: Vygeneruj 89 rôznych chybových prúdových kľúčov prislúchajúcich jedno-bitovým indukovaným chybám
3: Vyhodnoť prúdové kľúče na zistenie 2 bitov $LFSR_C$
4: Opakuj kroky 2., 3. pri rôznych posunoch šifry, kým nezískaš 39 lineárne nezávislých rovníc nad GF(2) nad pôvodným naplnením $LFSR_C$.
5: Pomocou známeho naplnenia časového registra zisti počiatočné naplnenie dátového registra $LFSR_D$.

3.2.3 Nový chybový útok na LILI-128

Náš útok sa líši od útoku popísaného v časti 3.2.2 tým, že na zistenie naplnenia časového registra nepožaduje indukovanie chýb v dátovom registri. Miesto toho sa pri hľadaní naplnenia časového registra využije útok fázovým posunom. Vygenerujeme bezchybový prúdový kľúč, „zresetujeme“ zariadenie, posunieme dátový register o jeden takt a vygenerujeme chybový prúdový kľúč. Toto zopakujeme s posunom o dva, tri a štyri takty.

Označme z_i i-ty bit bezchybového prúdového kľúča a $\hat{z}_i$ i-ty bit chybového prúdového kľúča. Porovnávaním bezchybového prúdového kľúča a chybových prúdových kľúčov dokážeme zistiť, o aký počet taktov bol posunutý dátový register pri generovaní jednotlivých bitov prúdového kľúča. Ak totiž platí, že sa nezhoduje bit bezchybového prúdového kľúča z_i a bit $\hat{z}_{i-1}$ chybového prúdového kľúča zodpovedajúceho dátovému registru posunutého o 1 takt, musí platiť, že v danom mieste muselo prísť ku posunu o 2, 3 alebo 4 takty. Preto porovnáme, či sa zhoduje bit bezchybového prúdového kľúča z_i a bit $\hat{z}_{i-1}$ prúdového kľúča zodpovedajúceho dátovému registru posunutého o 2 takty, a ak nie, znamená to, že muselo dôjsť k posunu o 3 alebo 4 takty. Preto porovnáme, či sa zhoduje bit bezchybového prúdového kľúča z_i a bit $\hat{z}_{i-1}$ prúdového kľúča zodpovedajúceho dátovému registru posunutého o 3 takty a ak nie, vieme, že muselo dôjsť ku posunu o 4 takty. Analogicky dokážeme ur-

čit', či došlo k posunu o 1, 2 alebo 3 takty. Tak zistíme príslušné bity v časovom registri.

Po zistení počiatočného naplnenia časového registra postupujeme pri hľadaní počiatočného naplnenia dátového registra pomocou indukcie 89 jedno-bitových chýb v dátovom registri (bez počiatočného fázového posunu). Pomocou algoritmu pre nájdenie počiatočného naplnenia dátového registra [23] potom zistíme toto naplnenie. Algoritmicky sa náš útok dá zapísať nasledovne:

Algoritmus 5: Chybový útok na LILI-128 (Hromada, Vojvoda)

1: Vygeneruj bezchybový prúdový kľúč
2: Vygeneruj 4 chybové prúdové kľúče, ktoré zodpovedajú fázovým posunom dátového registra o 1, 2, 3, 4 takty.
3: Analyzuj chybové prúdové kľúče na zistenie jednotlivých bitov časového registra a získanie 39 lineárne nezávislých rovníc nad GF(2) nad počiatočným naplnením časového registra.
4: Pomocou známeho naplnenia časového registra indukovaním 89 chýb v dátovom registri zisti počiatočné naplnenie dátového registra $LFSR_D$.

3.2.4 Výsledky experimentov

Náš útok sme implementovali v jazyku C. Sledovali sme, aká bude úspešnosť tohto útoku pri danej veľkosti prúdového kľúča (t.j. koľko útokov zo 100 vykonaných bude úspešných), kde použité počiatočné naplnenia boli generované náhodne.

V tabuľke 3.4 uvádzame výsledky. Riadok „$LFSR_C$ nenájdený“ obsahuje počet prípadov, kde nedošlo k úspešnému nájdeniu počiatočného naplnenia časového registra. Dôvodom je krátky prúdový kľúč.

Dĺžka prúd. kľúča	128b	160b	192b	224b	256b	288b	320b	352b
Úspešné útoky	0	5	20	63	71	90	96	100
Neúspešné útoky	100	95	80	37	29	10	4	0
$LFSR_C$ nenájdený	100	95	80	37	29	10	4	0

Tabuľka 3.4: Chybový útok na LILI-128 (Hromada)

Navyše sme vykonali 100 útokov bez obmedzenia veľkosti prúdového kľúča a zisťovali sme, aká bola minimálna, maximálna a priemerná hodnota potrebného počtu bitov prúdového kľúča na ich úspešné vykonanie. Zistili sme nasledovné hodnoty:

- Minimálny počet bitov chybového kľúča potrebný na úspešný útok: 140
- Maximálny počet bitov chybového kľúča potrebný na úspešný útok: 350
- Priemerný počet bitov chybového kľúča potrebný na úspešný útok: 200

Pre porovnanie sme implementovali aj pôvodný chybový útok, popísaný v časti 3.2.2. Opäť sme sledovali, aká bude úspešnosť tohto útoku pri danej veľkosti prúdového kľúča (t.j. koľko útokov zo 100 vykonaných bude úspešných) a vykonali 100 útokov bez obmedzenia veľkosti prúdového kľúča a sledovali, aká bola minimálna, maximálna a priemerná hodnota potrebného počtu bitov prúdového kľúča na ich úspešné vykonanie.

V tabuľke 3.5 uvádzame výsledky.

	30b	40b	50b	60b	70b
Úspešné útoky	0	0	72	99	100
Neúspešné útoky	100	100	28	1	0
$LFSR_C$ nenájdený	98	56	2	0	0
$LFSR_C$ nájdený zle	2	40	13	1	0
$LFSR_D$ nenájdený	0	4	13	0	0

Tabuľka 3.5: Chybový útok na LILI-128 (Hoch, Shamir)

Zo série 100 chybových útokov sme zistili nasledovné hodnoty:

- Minimálny počet bitov chybového kľúča potrebný na úspešný útok: 43
- Maximálny počet bitov chybového kľúča potrebný na úspešný útok: 65
- Priemerný počet bitov chybového kľúča potrebný na úspešný útok: 49

3.2.5 Záver

Oba útoky boli implementované za predpokladu, že viem chybu spoľahlivo identifikovať, t.j. lokalizovať každú indukovanú chybu na prvý krát a navyše pri každej indukcii chyby dôjde k jej indukcii na inom mieste registra. Za tohto predpokladu je vidieť, že pôvodný útok vyžaduje menší počet bitov prúdového kľúča. Avšak, v praxi na spoľahlivú identifikáciu chyby potrebujeme mať k dispozícii v priemere $\log \binom{n}{k} \cdot 2^{t+1}$ bitov prúdového kľúča, čo by postačovalo aj nášmu útoku. Preto je náš útok ekvivalentný s pôvodným z hľadiska počtu bitov prúdového kľúča potrebných na jeho úspešné vykonanie.

Rozdiel týchto dvoch útokov spočíva v hľadaní časového registra. Pôvodný útok používa indukciu chýb v dátovom registri pri rôznych posunoch časového registra, náš útok využíva útok fázovým posunom. Tu spočíva výhoda nášho útoku - za predpokladu, že je útok fázovým posunom možný, potrebuje v priemere 20-krát menší počet indukovaných chýb. Tento fakt je výhodou, keďže indukcie chýb v zariadeniach so sebou nesú riziko jeho poškodenia, preto menší počet indukovaných chýb znamená aj menšiu pravdepodobnosť jeho poškodenia.

Kapitola 4

Návrhy prúdových šifier

V tejto kapitole sú uvedené 2 články na tému návrhu nových prúdových šifier.

4.1 Prúdová šifra založená na šifre Fialka M-125

Prvý článok v tejto kapitole sa zaoberá návrhom novej prúdovej šifry, ktorej dizajn vychádza z klasickej šifry používanej počas studenej vojny - sovietskeho šifrátora Fialka M-125. Fialka je dodnes považovaná za nezlomenú, a teda bezpečnú šifru. Vytvorili sme návrh prúdovej šifry, ktorej konštrukcia vychádza z dizajnu Fialky - 10 nepravidelne taktovaných rotorov pracujúcich v 2 nezávislých módoch a návrh sme podrobili štatistickým testom - celkovo sme testovali 3 varianty s rôznymi veľkosťami bloku: 4-bity, 5-bitov a 8-bitov. Následne sme navrhli zjednodušenia (menší počet kôl, výmena modulárnych operácií za XOR) a zistili sme, že najrýchlejšia konštrukcia, ktorá prechádza testami je 8-bitová verzia so 6 rotormi využívajúca alebo modulárne operácie, alebo operáciu XOR.

4.1.1 Úvod

Ešte pred pár rokmi sa dala kryptografia rozdeliť do dvoch disjunktných častí: klasická kryptografia (šifry a šifrovacie zariadenia používané pred 2. svetovou vojnou) a moderná kryptografia (šifry a šifrovacie zariadenia používané po 2. svetovej vojne). Avšak v posledných rokoch dochádza k novému trendu - dizajn moderných šifier je ovplyvnený starými „klasickými“ šiframi, ktoré sú i dnes považované za bezpečné. Jednou z takýchto šifier je šifra Hummingbird [17]. Jej dizajn, kombinujúci blokové a prúdové šifry, je založený na slávnej nemeckej šifre Enigma. Tento zaujímavý trend nás priviedol na myšlienku navrhnúť prúdovú šifru založenú na inej starej, ale stále bezpečnej, šifre Fialka [43].

Prúdové šifry sú šifry, v ktorých sa šifrovacia/dešifrovacia transformácia mení s každým spracovaným znakom. Dá sa povedať, že majú „pamäť“, pretože zvyčajne každý nasledujúci stav šifry záleží alebo na predchádzajúcich stavoch šifry, alebo

na predchádzajúcom výstupe šifry. Väčšinou sú konštruované ako binárne aditívne šifrátory, v ktorých sa výstup pseudonáhodného generátora prixoruje k otvorenému textu. Sú považované za rýchlejšie a menej hardvérovo náročné, ako blokové šifry, čo ich robí vhodnými kandidátmi pre použitie v zariadeniach s obmedzenou výpočtovou silou a taktiež pre tzv. lightweight kryptografiu. O ich aktuálnosti svedčí aj to, že v roku 2008 skončil projekt eSTREAM [44, 1] zameraný na nájdenie nových prúdových šifier použiteľných či už v hardvéri, alebo v softvéri.

Sovietsky šifrovací stroj Fialka [43] je príkladom mechanického šifrovacieho zariadenia používaného v studenej vojne. Jeho dizajn pripomína prúdovú šifru, keďže každé písmenko otvoreného textu je šifrované inou transformáciou, v závislosti na predchádzajúcej šifrovacej transformácii (inými slovami, 2 rovnaké písmená na 2 rôznych pozíciách sa s veľkou pravdepodobnosťou zašifrujú na 2 rôzne písmená). Konštrukcia Fialky využíva množinu rotorov a fixných permutácií, na druhej strane, moderné prúdové šifry využívajú lineárne a nelineárne spätnoväzobné posuvné registre, nelineárne filtrovacie funkcie a nepravidelné taktovanie registrov.

Tento rozdiel v konštrukcii Fialky a moderných prúdových šifier nás inšpiroval ku návrhu prúdovej šifry, ktorý by vychádzal z konštrukcie Fialky. Navrhli sme niekoľko variant tejto šifry s rôznymi veľkosťami vnútorného stavu a rôznym počtom kôl. Následne sme tieto varianty podrobili štatistickým a výkonnostným testom.

4.1.2 Dizajn šifry Fialka M-125

Fialka M-125 je elektro-mechanický rotorový šifrátor (Obr. 4.1) [43] vytvorený sovietskou armádou. Používať sa začal okolo roku 1965 a oficiálne bol používaný až do pádu Sovietskeho zväzu v roku 1991, pričom bol taktiež používaný aj v ČSSR. Dizajn šifry je z veľkej časti založený na známej šifre Enigma, avšak niekoľko väčších odlišností naznačuje snahu o odstránenie známych nedostatkov Enigmy. Medzi hlavné vylepšenia patria: [43]

- Počet rotorov bol zvýšený na 10.
- Jednoduché krokovanie rotoru Enigmy bolo nahradené pokročilým dvojsmerným krokovaním závislým od blokovacieho pinu.
- Každé vstupné písmeno môže byť zašifrované na seba.

V tejto práci neuvádzame úplný šifrovací mechanizmus Fialky. Taktiež sa nevenujeme všetkým komponentom do hĺbky. Zameriavame sa len na niektoré časti nevyhnutné pre zachovanie hlavného princípu práce šifry. Pre úplný technický a elektronický popis Fialky, pozri [43] a webovú stránku http://www.cryptomuseum.com/crypto/fialka/index.htm ktorá taktiež obsahuje simulátory šifrovania. Popis a matematický model šifrovania sa dá nájsť v práci [4].

Obr. 4.1: Šifrovací stroj Fialka [41]

4.1.3 Konštrukcia šifry

V tejto časti popíšeme hlavné mechanické časti šifry Fialka.
Mechanický šifrátor Fialka M-125 sa dá podľa funkcionality rozdeliť na niekoľko častí:

- **Klávesnica** obsahuje 30 kláves slúžiacich pre zadávanie vstupu. Stlačenie klávesy pošle elektrický signál do výstupu zariadenia cez niekoľko elektro-mechanických častí popísaných nižšie.
- **Čítačka karty** umožňujúca používať dierne karty reprezentujúce fixné permutácie vstupnej abecedy. Taktiež bolo možné používať kovový trojuholník reprezentujúci identitu.
- Jadro šifrátora pozostáva z 10 **rotorov**. Po zašifrovaní písmena sa rotory posunú do novej pozície, takže v ďalšom kole je realizovaná iná permutácia vstupnej abecedy.
- Množina 10 rotorov je pripojená k statickým častiam zariadenia, **vstupnému disku** (sprava) a **reflektoru** (zľava). Tieto 2 časti majú podobnú štruktúru ako rotory. Každý z nich obsahuje 30 kontaktov. Reflektor zabezpečuje vrátenie signálu späť do rotorov. Má špeciálne zapojenie umožňujúce zašifrovanie vstupného písmena na to isté písmeno. Z tohto dôvodu sa musia používať 2 samostatné módy pre šifrovanie a dešifrovanie.

Šifrovací proces je založený na elektro-mechanických princípoch. Šifrovanie vstupného písmena vykonáva zložitý elektronický obvod. Príslušné mechanické časti zmenia nastavenie šifry (posun rotorov do novej pozície). Nastavenie elektronického obvodu závisí na počiatočnom nastavení zariadenia a mení sa po každom zašifrovanom znaku. Výstupné písmeno sa môže alebo vytlačiť na pásku, alebo vyraziť použitím 5-bitového kódu. Šifrovací proces je zhrnutý v algoritme 6.

Algoritmus 6: Šifrovacia procedúra vo Fialke

Vstup: Tajný kľúč K

```
1:  Nastav počiatočné nastavenie šifry podľa tajného kľúča K.
2:  for all stlačené klávesy na klávesnici do
3:     substitúcia z klávesnice do čítačky kariet;
4:     substitúcia z čítačky kariet do vstupného disku;
5:     for i:=1 to 10  do
6:        substitúcia pomocou rotora;
7:     end for
8:     substitúcia pomocou reflektora;
9:     for i:=10 to 1 do
10:       substitúcia pomocou inverzného rotora;
11:    end for
12:    inverzná substitúcia z čítačky kariet do vstupného disku;
13:    inverzná substitúcia z klávesnice do čítačky kariet;
14:    vytlač výstup;
15:    posuň rotory;
16: end for
```

Šifrovanie vieme popísať aj matematicky [4]. Definujme zobrazenie abecedy otvoreného textu a abecedy zašifrovaného textu do okruhu $\mathbb{Z}_N = \{0, 1, \ldots, N-1\}$, kde N je počet znakov (v pôvodnej Fialke $N = 30$). V každom kroku šifrovania sa vykonajú nasledovné operácie:

1. jedno písmeno $x \in \mathbb{Z}_N$ sa zašifruje

$$y = Enc(x) = \left(IP^{-1} \circ ROT^{-1}[t] \circ REF \circ ROT[t] \circ IP\right)(x);$$

   kde IP je počiatočná permutácia daná substitúciami z klávesnice/čítačky kariet/vstupného disku, REF je reflektor, ROT je permutácia daná rotormi a t reprezentuje stav šifry závislý od času (každého spracovaného znaku);

2. aktualizuje sa vnútorný stav zariadenia, rotory sa posunú do nových pozícií.

Permutácia ROT v $2l$-rotorov zariadení (2 nezávislé časti, každá s l rotormi - viď časť 4.1.3) je daná [4] ako:

$$ROT[t] = \sigma^{(l)}[t] \circ \rho^{(1)}[t] \cdots \sigma^{(2)}[t] \circ \rho^{(l-1)}[t] \circ \sigma^{(1)}[t] \circ \rho^{(l)}[t], \tag{4.1}$$

kde $\rho^{(i)}$ sú rotory v smere hodinových ručičiek a $\sigma^{(i)}$ sú rotory v protismere hodinových ručičiek (takže $\sigma^{(l)}$ je rotor číslo 1, $\rho^{(1)}$ je rotor číslo 2, atď.).

Rotory

Jadro šifrovania vo Fialke je realizované rotormi. Rotor je valec [20] z nevodivého materiálu, ktorého vrchná a spodná podstava obsahujú elektrické kontakty, reprezentujúce vstup a výstup rotora. Vo vnútri valca sú tieto kontakty vzájomne prepojené, čo realizuje fixnú permutáciu 30 vstupno/výstupných pozícií. Aktuálne používaná permutácia vstupnej abecedy taktiež závisí na relatívnej pozícii rotora [43].

Známa je existencia 2 typov rotorov: fixné rotory PROTON-1 a nastaviteľné rotory PROTON-2. Jadro nastaviteľných rotorov obsahujúce kontakty a prepojenia mohlo byť vymeniteľné medzi rôznymi rotormi, prípadne zmenené priamo v jednom rotore, čo má za následok zväčšenie priestoru kľúčov. V tejto práci sa zameriame na štandardné rotory PROTON-1.

Rotory sa delia do dvoch nezávislých skupín: jedna skupina sa točí v smere hodinových ručičiek a druhá skupina v opačnom smere (podrobnejšie v časti o taktovaní rotorov). Nech S je permutácia realizovaná ľubovoľným rotorom v základnej pozícii. Ak sa tento rotor otáča v smere hodinových ručičiek a v čase t sa posunie o $c[t]$ krokov zo základnej pozície, potom permutácia $\rho[t]$ sa dá vyjadriť ako [4]:

$$\rho[t] = S(x + c[t]) - c[t], \tag{4.2}$$

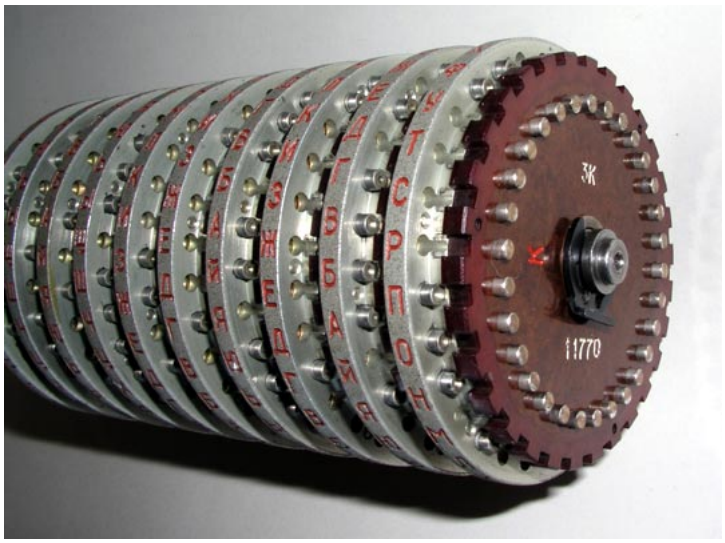
kde operácie $+, -$ sú v $\mathbb{Z}_N$ (modulo N). Podobne, pre rotor otáčajúci sa v opačnom smere [4]:

$$\sigma[t] = S(x - c[t]) + c[t]. \tag{4.3}$$

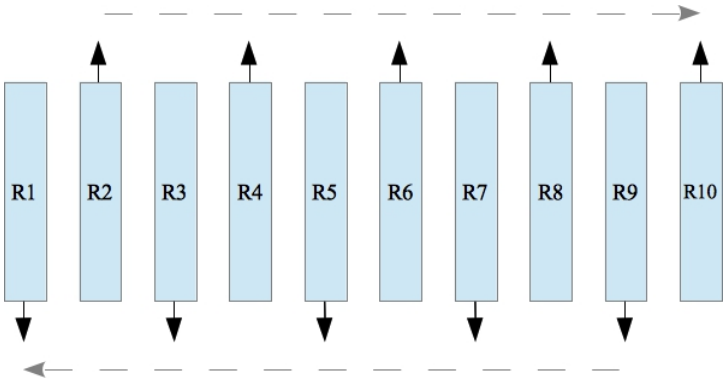
Taktovanie rotorov

Taktovanie (rotácia) rotorov je v prípade Fialky komplikovanejšia ako v prípade Enigmy. Ak označíme rotory Fialky číslami v poradí od 1 do 10, rotory označené nepárnymi číslami tvoria jednu nezávislú časť a zvyšné rotory tvoria druhú nezávislú časť (viď obrázok 4.3) [4].

Rotory v týchto 2 nezávislých častiach sa otáčajú v opačnom smere (obrázok 4.3). Každý rotor má po svojom obvode blokovacie piny (viď obrázok 4.2). Prítomnosť blokovacieho pinu na určitej pozícii bráni v rotácii všetkým nasledujúcim rotorom. V prípade rotora s párnym číslom blokuje tento blokovací pin všetky rotory s párnymi



Obr. 4.2: Množina 10 rotorov [41]



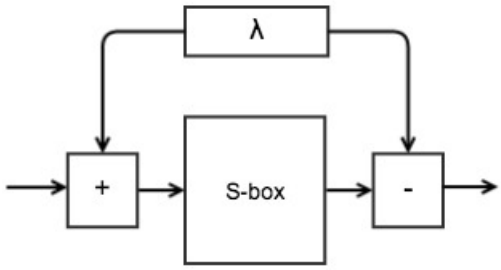
Obr. 4.3: Smer rotácie rotorov [4]

číslami napravo, v prípade rotora s nepárnym číslom blokuje tento pin všetky rotory s nepárnymi číslami naľavo [41, 4]. Prepojenie rotorov ilustruje prerušovaná čiara na obrázku 4.3.

Pozície rotorov ovplyvňujú permutácie jednotlivých rotorov podľa rovníc (4.2) a (4.3). Navyše, ovplyvňujú pozíciu blokovacieho pinu v danom čase, čo ďalej ovplyvňuje pozíciu rotorov v ďalšom kroku.

Rotory označené číslami 2 a 9 sa otočia v každom takte o 1 pozíciu, keďže nie sú ovládané žiadnym blokovacím pinom [43].

Uvažujme pre jednoduchosť len jednu množinu l rotorov (jednu nezávislú časť), ktorej krokovanie je ovládané blokovacími pinmi. Prvý rotor sa otáča voľne, t.j. v každom takte $c[t+1] = c[t] + 1 \mod N$. Pozíciu blokovacích pinov na rotore popíšeme poly-



Obr. 4.4: Návrh jedného rotora

nómom $a(x) \in \mathbb{Z}_2[x]$. Koeficient $a_i = 0$ ak je blokovací pin na i-tej pozícii a $a_i = 1$ ak blokovací pin na danej pozícii nie je. Základná pozícia má $i = 0$, ďalšia v smere otáčania má $i = 1$. Jeden krok rotora sa dá jednoducho zapísať ako násobenie polynómov, napríklad ak sa rotor posunie v čase t, dostávame [4]:

$$a(x)[t+1] = x \cdot a(x)[t] \mod (x^N + 1). \tag{4.4}$$

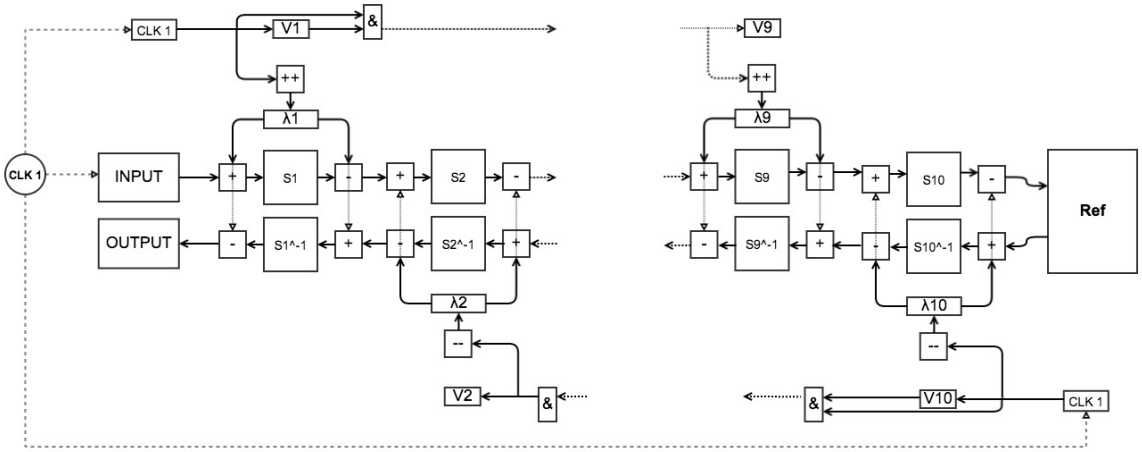
Ďalšie informácie o taktovaní rotorov sa dajú nájsť v článku [4].

4.1.4 Návrh prúdovej šifry

V časti 4.1.2 sme popísali hlavné princípy činnosti šifry Fialka. Jadrom šifrovacieho mechanizmu sú permutácia daná rotormi a taktovanie rotorov riadené blokovacími pinmi. V tejto časti popisujeme návrh prúdovej šifry založenej na týchto mechanizmoch. Je potrebné určiť, ako vhodne reprezentovať rotory a riadenie ich otáčania.

Permutácia daná rotorom sa dá vhodne reprezentovať pomocou jednoduchého S-boxu. Túto permutáciu S, vyjadrenú rovnicou (4.2), je možné vyjadriť obrázkom 4.4, kde λ je posun rotora a operácie $+, -$ sú definované v $\mathbb{Z}_N$ (modulo N). V tejto konštrukcii (viď obrázok 4.5) sa používa počítadlo λ ($c[t]$ v rovnici (4.2)) na otočenie rotora a modulárne sčítanie a odčítanie na zmenu permutácie rotora.

Taktovanie rotorov podľa časti 4.1.3 sa riadi blokovacími pinmi. V našom prípade sú piny navrhnuté ako binárne posuvné rotačné registre V_i na obrázku 4.5. Binárna hodnota na špecifickej pozícii (napr. bit najviac vpravo) ovláda posun ďalšieho rotora. Ak je prítomný blokovací pin (hodnota registra na špecifickej pozícii je 0) alebo je hodnota vstupného signálu z predchádzajúceho rotora 0, všetky ďalšie rotory (z danej množiny nezávislých rotorov) ostávajú stáť, t.j. žiadny z registrov V_i sa neposunie a hodnoty λ ostanú nezmenené.



Obr. 4.5: Návrh šifry (v skratke)

Šifra Fialka (podľa obrázka 4.5) sa dá zostrojiť použitím nasledovných operácií (v zátvorkách uvádzame označenie na obrázku 4.5):

- Substitúcia S-boxami (S_i, Ref),
- Modulárne sčítanie a odčítanie $(+, -)$,
- Počítadlo (λ, $++, --$),
- Rotačný posuvný register (V_i),
- Binárny AND (&).

Ako vidno na obrázku 4.5 potrebujeme 8 posuvných registrov, V_2 a V_9 sa nepoužívajú, keďže tieto rotory neovládajú žiadne ďalšie rotory.

Šifra Fialka sa dá zostrojiť pomocou 8 posuvných registrov, 8 hradiel AND, 10 počítadiel, 10 S-boxov pre rotory s 10 inverznými S-boxami a 1 S-boxom pre reflektor s príslušným inverzným S-boxom.

Pre $n-$bitovú verziu šifry je veľkosť počítadla λ n-bitov, veľkosť jedného posuvného registra 2^n-bitov a veľkosť jedného S-boxu 2^n-bitov. Zašifrovanie n-bitového vektora $x \in \mathbb{Z}_2^n$ podľa časti 4.1.3 vyzerá nasledovne:

$$y = Enc(x) = \left(ROT^{-1}[t] \circ REF \circ ROT[t]\right)(x);$$

kde $y \in \mathbb{Z}_2^n$ reprezentuje zašifrovaný text, REF je reflektor realizovaný bijektívnym S-boxom n-bitov na n-bitov and ROT je permutácia daná rotormi. Vynechali sme permutáciu IP a jej inverziu, pretože neprispieva k bezpečnosti šifry.

Permutácia ROT je daná

$$ROT[t] = \sigma^{(l)}[t] \circ \rho^{(1)}[t] \cdots \sigma^{(2)}[t] \circ \rho^{(l-1)}[t] \circ \sigma^{(1)}[t] \circ \rho^{(l)}[t], \tag{4.5}$$

kde l je počet rotorov v nezávislej časti, $\rho^{(i)}$ a $\sigma^{(i)}$ reprezentujú rotorové permutácie realizované bijektívnymi S-boxami a t predstavuje stav šifry - stav posuvných registrov V_i. Dešifrovanie:

$$x = Dec(y) = \left(ROT^{-1}[t] \circ REF^{-1} \circ ROT[t]\right)(y);$$

kde $x \in \mathbb{Z}_2^n$ predstavuje otvorený text, $y \in \mathbb{Z}_2^n$ predstavuje zašifrovaný text, REF^{-1} je realizovaný inverzným S-boxom ku REF a ROT je permutácia daná rotormi.

Najvyššia možná perióda šifry Fialka [4] je N^l, kde l je počet rotorov. Perióda taktovania systému l rotorov, s (w_1, w_2, …, w_{l-1}, w_l) blokovacími pinmi na jednotlivých rotoroch je (bez ohľadu na pozíciu pinov):

$$T_l = \frac{N^l}{\prod_{i=1}^{l-1} \gcd(N - w_i, N)}. \tag{4.6}$$

V nasledujúcich častiach popíšeme 3 rôzne verzie šifry, 4-bitovú, 5-bitovú a 8-bitovú konštrukciu. Všetky tieto verzie sú založené na konštrukcii prezentovanej na obrázku B.1, ktorý sa nachádza v Dodatku B tejto práce.

4-bitová konštrukcia

Implementovali sme 4-bitovú verziu šifry, $n = 4$, t.j. vstup a výstup šifry sú 4 bity podľa dizajnu popísaného v časti 4.1.4. Použili sme S-boxy, ktoré sú odolné voči lineárnej a diferenciálnej kryptoanalýze, z [54].
Perióda tejto 4-bitovej verzie je 2^{20} [4].
Tajným kľúčom sú inicializačné hodnoty registrov V_i a počiatočné hodnoty λ_i.
Zhrnutie 4-bitovej konštrukcie:

- 11 S-boxov $\mathbb{Z}_2^4 \to \mathbb{Z}_2^4$ s príslušnými inverziami.
- Veľkosť kľúča: 200 bitov
- Perióda: 2^{20}

Veľkosť kľúča je redukovateľná na 192 bitov, ak ignorujeme registre V_2 a V_9.

5-bitová konštrukcia

5-bitová verzia, $n = 5$, je podobná verzii z časti 4.1.4.

- 11 S-boxov $\mathbb{Z}_2^5 \to \mathbb{Z}_2^5$ s príslušnými inverziami.

- Veľkosť kľúča: 370 bitov
- Perióda: 2^{25}

Aj v tomto prípade môže byť veľkosť kľúča redukovateľná - o 10 bitov. Problémom softvérovej implementácie je, že vstup treba deliť na pätice bitov, t.j. optimálnym vstupom na 8-bitových procesoroch je 40 bitov.

8-bitová konštrukcia

Zväčšenie bitovej veľkosti šifry má za následok zväčšenie vnútorného stavu. V prípade 8-bitovej konštrukcie znázornenej na obrázku B.1 potrebujeme desať S-boxov $\mathbb{Z}_2^8 \to \mathbb{Z}_2^8$. Rotorové S-boxy zaberajú 20480 bitov, k nim inverzné taktiež. S-box pre reflektor spolu s inverziou zaberajú 4096 bitov. Posuvné registre zaberajú 2560 bitov pre všetky rotory. Počítadlo zaberie 80 bitov.

Aby sme zmenšili veľkosť vnútorného stavu, budeme pre všetky rotory používať jeden S-box spolu s jeho inverziou. Ostatné časti ostanú bez zmeny. To zredukuje veľkosť vnútorného stavu o **36864 bitov**. Taktiež môžeme redukovať priestor kľúčov. Vynechaním naplnení registrov z kľúča šifry, t.j. zafixovaním registrov, sa nám zníži veľkosť kľúča na 80 bitov - obsah počítadiel. Ak chceme zvýšiť veľkosť kľúča na 160 bitov, môžeme použiť 80 bitov pre určenie posunov 10 registrov z počiatočnej pozície.

8-bitová verzia šifry, $n = 8$:

- 2 S-boxy $\mathbb{Z}_2^8 \to \mathbb{Z}_2^8$ s príslušnými inverziami.
- Veľkosť kľúča: 80 bitov (alebo 160 bitov).
- Perióda: 2^{40}.
- Vnútorný stav: 10832 bitov.

Referenčná implementácia 8-bitovej verzie sa nachádza v Dodatku A tejto práce.

4.1.5 Analýza návrhu

Rýchlosť softvérovej implementácie šifry sme analyzovali na počítači s parametrami: CPU 2.8 GHz Intel Core i7, RAM 4GB 1333 MHz DDR3, Mac OS X 10.7.5.

- 4-bitová verzia - 61 $Mbit/s$−367 $cyklov/B$
- 5-bitová verzia - 82 $Mbit/s$−273 $cyklov/B$
- 8-bitová verzia - 124 $Mbit/s$−180 $cyklov/B$

Taktiež sme vykonali štatistické testy nášho návrhu. S každou verziou šifry sme vygenerovali sto 1MB postupností a aplikovali sme sadu štatistických testov NIST [45] na hladine významnosti $\alpha = 0.01$. Verzia šifry daným testom prešla, ak ním prešlo aspoň 96 postupností z príslušnej množiny. Výsledky testovania sú prezentované v tabuľkách uvedených nižšie.

Ďalej sme sa pokúsili o trade-off medzi rýchlosťou a bezpečnosťou znížením počtu rotorov a nahradením modulárneho sčítania a odčítania operáciou XOR. Zníženie počtu rotorov spôsobilo zmenšenie veľkosti kľúča, 4-bitová a 5-bitová verzia s $2l$ rotormi má veľkosť kľúča $2l * n + 2l * 2^n$. V prípade 8-bitovej verzie je veľkosť kľúča $2l * n$ (alebo $2l * n * 2$).

V nasledujúcich tabuľkách každý stĺpec predstavuje jeden typ šifry - prvý riadok obsahuje počet rotorov a použitú operáciu („mod“ znamená modulárne sčítanie/odčítanie, „xor“ znamená XOR), druhý riadok obsahuje priemernú rýchlosť šifrovania meranú v megabitoch za sekundu a tretí riadok obsahuje zhrnutie, či verzia šifry prešla alebo neprešla štatistickým testom.

Výsledky 4-bitovej verzie:

Rotory / operácia	10/mod	10/xor	8/mod	8/xor
Rýchlosť $Mbit/sec$	61	80	72	92
NIST testy	ÁNO	NIE	NIE	NIE

Tabuľka 4.1: Výsledky 4-bitovej verzie

Tie isté experimenty sme vykonali aj na 5-bitovej verzii. Štatistickými testami neprešla len 8-rotorová verzia s operáciou XOR. Je dôležité poznamenať, že 5-bitová 8-rotorová verzia má tú istú výstupnú periódu ako 4-bitová 10-rotorová verzia.

Výsledky 5-bitovej verzie:

Rotory / operácia	10/mod	10/xor	8/mod	8/xor
Rýchlosť $Mbit/sec$	78	83	113	136
NIST testy	ÁNO	ÁNO	ÁNO	NIE

Tabuľka 4.2: Výsledky 5-bitovej verzie

Výsledky 8-bitovej verzie:

Rotory / operácia	10/mod	10/xor	8/mod	8/xor	6/mod	6/xor
Rýchlosť *Mbit/sec*	124	134	155	159	200	208
NIST testy	ÁNO	ÁNO	ÁNO	ÁNO	ÁNO	ÁNO

Tabuľka 4.3: Výsledky 8-bitovej verzie

Najrýchlejšia konštrukcia, ktorá prešla štatistické testy je 8-bitová verzia so 6 rotormi a operáciou XOR. Jej rýchlosť šifrovania je 208 Mbps, čo zodpovedá cca 107 cyklov na jeden bajt. Porovnali sme tieto výsledky s kandidátmi z projektu eSTREAM. Žiaľ, náš dizajn bol pomalší ako väčšina prúdových šifier z eSTREAMu určených pre softvérové použitie. Pre porovnanie:

- HC-128 - 4 cykly na bajt.
- Rabbit - 10 cyklov na bajt.
- Salsa20/12 - 11 cyklov na bajt.
- SOSEMANUK - 6 cyklov na bajt.

Taktiež sme porovnali rýchlosť nášho návrhu s rýchlosťou lightweight šifry Hummingbird [17] spomenutej v úvode 4.1.1 Výsledná rýchlosť bola 165 Mbps, čo naznačuje, že náš dizajn je rýchlejší, keďže sme dosiahli rýchlosť 208 Mbps. Avšak, toto porovnanie je len približné, keďže Hummingbird je hardvérovo orientovaná šifra, s implementáciami v FPGA, zatiaľ čo nás dizajn sme testovali ako program napísaný v jazyku C.

Bezpečnosť našej šifry vzťahujeme na bezpečnosť šifry Fialka - kým nie je nájdený úspešný útok na šifru Fialka považujeme náš dizajn za bezpečný. Navyše sme zvolili S-boxy, ktoré majú dobré bezpečnostné vlastnosti vzhľadom na lineárnu a diferenciálnu kryptoanalýzu [54]. Samozrejme, taktiež je možné použiť v podstate ľubovoľné S-boxy, napr. S-box zo šifry AES. Samozrejme, náš návrh je potrebné podrobiť ďalšej podrobnej kryptoanalýze, aby bolo možné tvrdiť, že je bezpečný pre bežné použitie, keďže prejdenie štatistickými testami je len nutnou podmienkou, aby bol považovaný za bezpečný.

4.1.6 Záver

Predstavili konštrukciu prúdovej šifry založenú na sovietskom šifrovacom algoritme Fialka. Vykonali sme testy rýchlosti a štatistické testovanie 3 verzií našej konštrukcie - 4-bitovej verzie, 5-bitovej verzie a 8-bitovej verzie. Preskúmali sme počet rotorov, potrebný pre úspešné prejdenie štatistickými testami NIST. Naše výsledky naznačujú, že 4-bitová verzia s 10 rotormi a modulárnymi operáciami týmito testami prechádza. Avšak akákoľvek modifikácia (XOR operácia, menší počet kôl) nie. Obe 5-bitové verzie s 10 rotormi testami prechádzajú a taktiež 8-rotorová verzia s modulárnymi

operáciami. 8-bitová verzia sa dá redukovať ešte viac - na 6 rotorov, či už s modulárnymi operáciami, alebo operáciou XOR.

4.2 Použitie kryptosystému Poly-Dragon v generátore náhodných čísiel MSTg

Tento článok vznikol ako výsledok zadania, ktoré autor dizertačnej práce dostal na úvod svojho doktorandského štúdia. Cieľom bolo preskúmať možnosť využitia kryptosystému s verejným kľúčom Poly-Dragon v kombinácii s generátorom náhodných čísiel MSTg založeného na náhodných pokrytiach grupy, pričom prvky náhodných pokrytí boli generované pomocou systému Poly-Dragon. Následne bol výsledný dizajn podrobený štatistickým testom na otestovanie „náhodnosti“ výstupu. Testovali sme viacero verzií a zistili sme, že nová konštrukcia má podobné štatistické vlastnosti ako stará verzia. Navyše je bezpečnosť náhodných pokrytí podmienená post-kvantovým kryptosystémom Poly-Dragon.

4.2.1 Úvod

Existujú 2 základné typy generátorov náhodných čísiel - nedeterministické náhodné generátory a deterministické náhodné generátory.

- Nedeterministické generátory - sú v skutku náhodné, ich výstup sa nedá predikovať, ani zopakovať. Väčšinou využívajú rôzne prírodné zdroje náhodnosti - rádioaktívny rozpad, tepelné žiarenie, obsah hardvérových zásobníkov, atď. Ich výhodou je, že výstup je naozaj náhodný. Ich nevýhodou je, že nie sú vhodné pre generovanie veľkého množstva dát pre kryptografické účely (napr. sa nedajú použiť ako generátory prúdového kľúča) - sú pomalé a vyžadujú spracovanie (post-processing), resp. extrakciu náhodnosti.
- Deterministické generátory - označované ako pseudonáhodné generátory (PRNG z anglického „Pseudorandom Number Generator“). Ich výstup nie je náhodný, ale závisí od náhodnej vstupnej hodnoty - tzv. seedu, z ktorého je na základe príslušného deterministického algoritmu generovaný výstup generátora. Na tieto generátory sa aplikujú štatistické testy, ktoré naznačujú, či sa daný generátor (jeho výstup) správa podobne, ako nedeterministický generátor.

4.2.2 MSTg - generátor založená na náhodných pokrytiach konečných grúp

V roku 2011 bol predstavený pseudonáhodný generátor čísel s označením MSTg [35], ktorého konštrukcia využíva náhodné pokrytia konečných grúp a jednocestné funkcie. Nasledujúca teória je prevzatá z príslušného článku [35].

Definícia 7 *Nech $\mathcal{G}$ je konečná abstraktná grupa. Šírka grupy $\mathcal{G}$ je definovaná ako kladné celé číslo $w = \lceil \log |\mathcal{G}| \rceil$.*

Definícia 8 *Nech $\mathcal{G}$ je konečná abstraktná grupa a $\mathcal{S}$ je podmnožina $\mathcal{G}$. Nech $\alpha = [\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_S]$ je usporiadaný súbor podmnožín $\mathcal{A}_i \in \mathcal{G}$ taký, že $\sum_{i=1}^{s} |\mathcal{A}_i|$ je ohraničená polynómom v šírke $\mathcal{G}$. Potom α nazývame pokrytím $\mathcal{S}$ ak pre každý prvok $h \in \mathcal{S}$ platí, že sa dá vyjadriť ako súčin tvaru*

$$h = g_1 \cdot g_2 \cdot ... \cdot g_s, \tag{4.7}$$

kde $g_i \in \mathcal{A}_i$ a $\cdot$ je grupová operácia definovaná na grupe $\mathcal{G}$.

Ak sú prvky pokrytia α (t.j. prvky podmnožín $\mathcal{A}_i$) generované náhodne (t.j. náhodne vyberané z príslušnej grupy), potom hovoríme, že α je *náhodné pokrytie*. Ak $\mathcal{G} = \mathcal{S}$, α sa nazýva pokrytím grupy $\mathcal{G}$.

Podmnožiny $\mathcal{A}_i$ sa nazývajú *bloky* a vektor $(r_1, r_2, ..., r_s)$, kde $r_i = |\mathcal{A}_i|$ sa nazýva *typ* pokrytia α. Pokrytie α sa nazýva *faktorizovateľné*, ak je možné vykonať faktorizáciu v (4.7) v polynomickom čase $\mathcal{O}(w)$ pre skoro všetky prvky $\mathcal{G}$, kde w je šírka grupy $\mathcal{G}$. V opačnom prípade sa pokrytie nazýva nefaktorizovateľné.

Vo všeobecnosti je problém nájdenia faktorizácie prvku grupy vzhľadom na náhodne vygenerované pokrytie α výpočtovo náročný. Napríklad, nech $\mathcal{G}$ je cyklická grupa a $g \in \mathcal{G}$ jej generátor, potom sa prvky množín $\mathcal{A}_i$ dajú reprezentovať ako mocniny tohto generátora a nájdenie faktorizácie vedie na riešenie problému diskrétneho logaritmu. Ak je naozaj nájdenie rozkladu prvku výpočtovo náročné, potom sa dajú na základe náhodných pokrytí definovať funkcie, ktoré sa správajú ako tzv. jednocestné funkcie nasledovným spôsobom.

Nech $\alpha = [\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_s]$ je náhodné pokrytie typu $(r_1, r_2, ..., r_s)$ grupy $\mathcal{G}$, kde $\mathcal{A}_i = [a_{i,1}, a_{i,2}, ..., a_{i,r_i}]$ a nech $m = \prod_{i=1}^{s} r_i$. Nech $m_1 = 1$ a $m_i = \prod_{j=1}^{i-1} r_j$ pre $j = 2, ..., s$. Nech τ označuje bijekciu z $\mathbb{Z}_{r_1} \oplus \mathbb{Z}_{r_2} \oplus ... \oplus \mathbb{Z}_{r_s}$ do $\mathbb{Z}_m$:

$$\tau : \mathbb{Z}_{r_1} \oplus \mathbb{Z}_{r_2} \oplus ... \oplus \mathbb{Z}_{r_s} \to \mathbb{Z}_m,$$

$$\tau(j_1, j_2, ..., j_s) := \textstyle\sum_{i=1}^{s} j_i m_i.$$

Použitím τ vieme definovať surjektívne zobrazenie $\breve{\alpha}$ na základe pokrytia α.

$$\breve{\alpha} : \mathbb{Z}_m \to \mathcal{G},$$

$$\breve{\alpha}(x) := a_{1,j_1} \cdot a_{2,j_2} \cdot ... \cdot a_{s,j_s},$$

kde $(j_1, j_2, ..., j_s) = \tau^{-1}(x)$. Keďže τ and τ^{-1} sú efektívne vypočítateľné, zobrazenie $\breve{\alpha}(x)$ je efektívne vypočítateľné.

Avšak, ak je dané pokrytie α a prvok $y \in \mathcal{G}$, určenie prvku $x = \breve{\alpha}^{-1}(y)$ nie je efektívne vypočítateľné, ak α nie je faktorizovateľné, t.j. ak nevieme v polynomickom čase určiť indexy $j_1, j_2, ..., j_s$ také, že $y = a_{1,j_1} \cdot a_{2,j_2} \cdot ... \cdot a_{s,j_s}$. Po získaní vektora $(j_1, j_2, ..., j_s)$ je ľahké vypočítať $\breve{\alpha}^{-1}(y) = \tau(j_1, j_2, ..., j_s)$. Preto, ak je α náhodné pokrytie veľkej podmnožiny $\mathcal{S}$ grupy $\mathcal{G}$, potom je nájdenie rozkladu (4.7) ťažký problém a zobrazenie

$$\breve{\alpha} : \mathbb{Z}_m \to \mathcal{S}$$

založené na α, kde $m = \prod_{i=1}^{s} |\mathcal{A}_i|$ je jednocestná funkcia.

Teraz popíšeme generátor MSTg tak, ako je navrhnutý v článku [35].

Podotýkame, že štruktúra grupy $\mathcal{G}$ je ľubovoľná. Nech $\mathcal{G}_1$ a $\mathcal{G}_2$ sú dve zvolené konečné grupy, kde $|\mathcal{G}_1| = n$ and $|\mathcal{G}_2| = m$ a $n \geq m$.

Nech ℓ je celé číslo také, že $\ell \geq n$. Nech $k \geq 0$ je pevne dané celé číslo.

Nech α je náhodné pokrytie typu $(u_1, u_2, ..., u_t)$ grupy $\mathcal{G}_1$, kde $\prod_{i=1}^{t} u_i = \ell$. Nech $\alpha_1, \alpha_2, ..., \alpha_k$ je množina náhodných pokrytí typu $(r_1, r_2, ..., r_s)$ grupy $\mathcal{G}_1$, kde $\prod_{i=1}^{s} r_i = |\mathcal{G}_1|$. Nech $\gamma = [H_1, H_2, ..., H_s]$ je náhodné pokrytie typu $(r_1, r_2, ..., r_s)$ grupy $\mathcal{G}_2$. Nech f_1 a f_2 sú bijektívne zobrazenia $f_1 : \mathcal{G}_1 \to \mathbb{Z}_n$ a $f_2 : \mathcal{G}_2 \to \mathbb{Z}_m$, ktoré zobrazia prvky grupy $\mathcal{G}_1$ na čísla z množiny $\mathbb{Z}_n$ a prvky grupy $\mathcal{G}_2$ na čísla z množiny $\mathbb{Z}_m$. Potom definujme funkciu:

$$F : \mathbb{Z}_\ell \to \mathbb{Z}_m$$

ako zloženie zobrazení (verzia $\mathcal{A}$):

$$\mathbb{Z}_\ell \xrightarrow{\breve{\alpha}} \mathcal{G}_1 \xrightarrow{f_1} \mathbb{Z}_n \xrightarrow{\breve{\alpha}_1} \mathcal{G}_1 \xrightarrow{f_1} \mathbb{Z}_n \longrightarrow ... \xrightarrow{\breve{\alpha}_k} \mathcal{G}_1 \xrightarrow{f_1} \mathbb{Z}_n \xrightarrow{\breve{\gamma}} \mathcal{G}_2 \xrightarrow{f_2} \mathbb{Z}_m.$$

Toto zobrazenie sa dá definovať alternatívnym spôsobom ako (verzia $\mathcal{B}$):

$$\mathbb{Z}_\ell \xrightarrow{\breve{\alpha}} \mathcal{G}_1 \xrightarrow{f_1} \mathbb{Z}_n \xrightarrow{\breve{\gamma}} \mathcal{G}_2 \xrightarrow{f_2} \mathbb{Z}_m \xrightarrow{\breve{\delta}_1} \mathcal{G}_2 \xrightarrow{f_2} \mathbb{Z}_m \longrightarrow ... \xrightarrow{\breve{\delta}_k} \mathcal{G}_2 \xrightarrow{f_2} \mathbb{Z}_m,$$

kde $\delta_1, \delta_2, ..., \delta_k$ je množina náhodných pokrytí typu $(v_1, v_2, ..., v_w)$ grupy $\mathcal{G}_2$ kde $\prod_{i=1}^{w} v_i = |\mathcal{G}_2|$.

Algoritmus 7 popisuje samotný generátor MSTg. Na generovanie výstupu používa jednoduchý tzv. *counter* mód (CTR mód, t.j. mód *počítadla*) - konštanta C sa v každej iterácii pripočíta k hodnote s_i. Jadrom generátora je funkcia F, t.j. výstup generátora, resp. jeho vlastnosti, závisia práve od pokrytí grúp $\mathcal{G}_1$ a $\mathcal{G}_2$. Inými slovami, na vygenerovanie týchto pokrytí je potrebné použiť iný dobrý pseudonáhodný generátor, napríklad Blum-Blum-Shubov generátor [8].

Algoritmus 7: MSTg: Pseudonáhodny generátor čísel založený na náhodných pokrytiach

Vstup : Celé čísla ℓ, m, funkcia $F : \mathbb{Z}_\ell \to \mathbb{Z}_m$ ako bola definovaná vyššie, náhodný a tajný seed $s_0 \in \mathbb{Z}_\ell$, konštanta $C \in \mathbb{Z}_m$

Výstup: t pseudonáhodných čísiel $z_1, z_2, ..., z_t \in \mathbb{Z}_m$

1. Pre i od 1 do t urob:

   1.1 $s_i = s_{i-1} + C \pmod{\ell}$

   1.2 $z_i = F(s_i)$

2. Vrát' $(z_1, z_2, ..., z_t)$

Štatistické vlastnosti MSTg

Náhodnosť výstupu pseudonáhodných generátorov, respektíve možnosť odlíšiť takýto výstup od výstupu nedeterministického generátora, sa zisťuje štatistickým testovaním. V skratke v tejto kapitole uvedieme štatistické testovanie generátora MSTg tak, ako ho prezentujú jeho autori v [35].

Na štatistické testovanie bol použitý balík štatistických testov NIST 800-22 (bližšie špecifikovaný v dokumente [45]). Tento balík pozostáva z 15 základných testov, pričom keďže niektoré testy sú parametrizovateľné, celkovo sa vykonáva 188 testovaní. Vstupom sú binárne postupnosti veľkosti 10^3 až 10^7 bitov, ich počet by mal byť väčší ako 55. Štatistické vyhodnocovanie sa deje na hladine významnosti 0.01.

Grupy, použité v generátore boli zvolené ako elementárne abelovské 2-grupy s grupovou operáciou XOR. Rád prvej grupy bol $|\mathcal{G}_1| = 2^{e_1}$, rád druhej grupy bol $|\mathcal{G}_2| = 2^{e_2}$. Išlo teda o grupy, ktorých prvky boli binárne vektory, ktorých dimenzie boli e_1, respektíve e_2. Pri takýchto grupách je jednoduché určiť bijekciu f_1, respektíve f_2, kde tieto sa zvolia ako identické zobrazenie binárneho čísla na jeho dekadickú reprezentáciu. Zvolená konštrukcia funkcie F bola verzia $\mathcal{A}$.

Postup pri testovaní generátora bol nasledovný:

1. Vygenerovala sa inštancia generátora MSTg, pričom na generovanie prvkov pokrytia sa použil generátor Blum-Blum-Shub.

2. Inštancia MSTg vygenerovala 100 binárnych postupností veľkosti 10^7 bitov. Na túto množinu postupností sa následne aplikovala sada štatistických testov NIST.

3. Tento proces sa zopakoval 1056 krát.

Následne autori podobným spôsobom testovali generátory Blum-Blum-Shub (BBS),

PRNG	Verzia	Výstup	$\#c$	C	f_{aver}	f_0/R	f_{max}	f_0	f_1	f_2	f_3	f_4	f_{5+}
MSTg	256/128	128	2	1	0.762	0.479	7	506	366	127	48	6	3
MSTg	256/128	128	3	1	0.798	0.450	5	475	389	131	53	7	1
MSTg	256/128	128	1	p_1	0.743	0.490	5	517	356	135	35	11	2
MSTg	256/128	128	2	p_1	0.686	0.514	6	543	354	116	37	4	2
MSTg	256/128	128	3	p_1	0.720	0.509	5	537	343	129	32	12	3
BBS	1024	1			0.742	0.467	7	493	392	129	37	4	1
ARC4	2048	8			0.690	0.513	4	542	331	154	26	3	0
MT	64	64			0.784	0.493	5	521	335	134	41	23	2

Tabuľka 4.4: NIST testovanie, prevzaté z [35]

ARC4 a Mersenne Twister (MT) generátor a porovnali dosiahnuté výsledky.

V tabuľke 4.4 sú prezentované dosiahnuté výsledky testovania pre jednotlivé generátory a ich rozdielne verzie. Nutno podotknúť, že jeden NIST test vráti množinu 188 p-hodnôt. Keďže bolo testovaných 1056 inštancií generátora, v konečnom dôsledku je dostupných pre 1 typ generátora 1056 množín p-hodnôt. Stĺpce f_i tabuľky, kde $i = 0, ..., 5+$ udávajú počty množín p-hodnôt, kde i p-hodnôt je menších ako požadovaná hladina významnosti 0.01. Stĺpec f_{max} udáva najvyšší počet p-hodnôt, ktoré nevyhovovali v jednej inštancii daného generátora, f_{aver} udáva priemerný počet nevyhovujúcich p-hodnôt na jedno testovanie. Stĺpec C udáva, či bolo ako aditívna konštanta v CTR móde použité číslo 1 alebo prvočíslo p_1 (256-bitové). Stĺpec $\#c$ udáva počet pokrytí použitých v danej verzii MSTg. Z tabuľky je vidno, že generátor MSTg dosahuje podobné výsledky ako ostatné generátory, teda aj ako generátor BBS, ktorý je považovaný za vhodný pre kryptografické účely. Vidno, že v priemere každá druhá inštancia generátora prešla testom NIST (t.j. množina p-hodnôt neobsahovala nevyhovujúce hodnoty).

4.2.3 Poly-Dragon - kryptosystém s verejným kľúčom

Generátor MSTg prezentovaný v časti 4.2.2 má flexibilnú štruktúru, keďže na generovanie náhodných pokrytí sa dá použiť ľubovoľný generátor, prípadne kryptosystém. My sme sa rozhodli na generovanie prvkov náhodných pokrytí použiť namiesto klasického generátora kryptosystém s verejným kľúčom Poly-Dragon.

Tento kryptosystém bol navrhnutý v roku 2010 v článku [49]. Jedná sa o kryptosystém, ktorého konštrukcia je založená na permutačných polynómoch nad konečnými poliami a ktorého bezpečnosť je založená na NP-úplnom probléme riešenia sústavy nelineárnych rovníc nad konečným poľom (MQ-problém). V tejto časti popíšeme tento kryptosystém tak, ako ho autori predstavili v ich článku. Začneme teoretickým úvodom do permutačných polynómov.

Definícia 9 *Nech p je prvočíslo, n je kladné celé číslo a $\mathbb{F}_q$ je konečné pole s $q = p^n$ prvkami. Polynóm $f(x) \in \mathbb{F}_q$ sa nazýva permutačný polynóm, ak je bijektívnym zobrazením pol'a $\mathbb{F}_q$ na pole $\mathbb{F}_q$.*

Inými slovami:

- Funkcia f je surjektívna.
- Funkcia f je injektívna.
- $f(x) = a$ má práve jedno riešenie v poli $\mathbb{F}_q$ pre každé $a \in \mathbb{F}_q$

Nech $B = \{\vartheta_0, \vartheta_1, ..., \vartheta_{n-1}\}$ je báza pol'a $\mathbb{F}_{2^n}$ nad pol'om $\mathbb{F}_2$. Každý prvok $x \in \mathbb{F}_{2^n}$ sa dá jednoznačné vyjadrit' ako $x = \sum_{i=0}^{n-1} x_i\vartheta_i$, kde $x_i \in \mathbb{F}_2$. Týmto spôsobom vieme jednoznačne previazat' polia $\mathbb{F}_{2^n}$ a $\mathbb{F}_2^n$ a prvok $x \in \mathbb{F}_{2^n}$ vyjadrit' ako n-ticu $(x_0, x_1, ..., x_{n-1})$ kde $x_i \in \mathbb{F}_2$. Hammingovu váhu prvku $x = (x_0, x_1, ..., x_{n-1})$ označíme ako $\omega(x)$, pričom je rovná počtu jednotiek x.

Definícia 10 *Nech $\mathbb{F}_{2^n}$ je konečné pole. Nech $B = (\vartheta_0, \vartheta_1, ..., \vartheta_{n-1})$ je báza tohoto pol'a nad $\mathbb{F}_2$. Nech prvok $\alpha \in \mathbb{F}_{2^n}$ pričom mu prislúcha n-tica $(\alpha_0, \alpha_1, ..., \alpha_{n-1})$. Potom definujeme linearizovaný polynóm (alebo p-polynóm) $L_\alpha(x)$ nad $\mathbb{F}_{2^n}$ prislúchajúci prvku $\alpha = (\alpha_0, \alpha_1, ..., \alpha_{n-1})$ ako:*

$$L_\alpha(x) = \sum_{i=0}^{n-1} \alpha_i x^{2^i}. \tag{4.8}$$

Taktiež, funkcia $Tr(x)$ označuje *stopu*, t.j. zobrazenie z pol'a $\mathbb{F}_{2^n}$ do pol'a $\mathbb{F}_2$ definované ako:

$$Tr(x) = x + x^2 + ... + x^{2^{n-1}}.$$

Ako bolo uvedené, konštrukcia kryptosystému Poly-Dragon využíva permutačné polynómy. Uvedieme teraz dve vety, ktoré platia pre permutačné polynómy nad poliami. Ich dôkazy sa dajú nájst' v článku [49].

Veta 1 *Nech n je nepárne kladné celé číslo a $\beta = (\beta_0, \beta_1, ..., \beta_{n-1})$ je prvok pol'a $\mathbb{F}_{2^n}$ taký, že $\omega(\beta)$ je párne a prvky 0 a 1 sú jediné korene $L_\beta(x)$ v poli $\mathbb{F}_{2^n}$. Nech k_1 a k_2 sú nezáporné celé čísla také, že $GCD(2^{k_1} + 2^{k_2}, 2^n - 1) = 1$. Nech ℓ je kladné celé číslo, pre ktoré platí $(2^{k_1} + 2^{k_2}) \cdot \ell \equiv 1 \pmod{2^n - 1}$ a γ je prvok pol'a $\mathbb{F}_{2^n}$, pričom platí $Tr(\gamma) = 1$. Potom*

$$f(x) = (L_\beta(x) + \gamma)^\ell + Tr(x)$$

je permutačný polynóm pol'a $\mathbb{F}_{2^n}$.

Veta 2 *Nech n je nepárne kladné celé číslo a α je prvok pol'a $\mathbb{F}_{2^n}$. Polynóm $g(x) = (x^{2^{k2^r}} + x^{2^r} + \alpha)^l + x$ je permutačný polynóm pol'a F_{2^n} ak $Tr(\alpha) = 1$ a $(2^{k2^r} + 2^r) \cdot l \equiv 1 \pmod{2^n - 1}$.*

Keďže Poly-Dragon je kryptosystém s verejným kľúčom, nasleduje popis generovania verejného kľúča. Pri generovaní sa využívajú oba permutačné polynómy $f(x)$ a $g(x)$.

Veta 1 aj Veta 2 obsahujú veľa voliteľných parametrov. Autori v článku [49] navrhujú nasledovnú voľbu parametrov. Nech $r = 0, n = 2m - 1, k = m, k_2 = m, k_1 = 0$, $l = 2^m - 1$ a $\ell = 2^m - 1$. V tomto prípade majú permutačné polynómy, použité na generovanie verejného kľúča, tvar $f(x) = (L_\beta(x) + \gamma)^{2^m-1} + Tr(x)$ a $g(x) = (x^{2^m} + x + \alpha)^{2^m-1} + x$. Parametre α, β, γ sú tajné parametre. Predpokladajme teraz, že s a t sú dve invertibilné afinné transformácie. Nech x označuje otvorený text a y zašifrovaný text. Vzťah otvoreného a zašifrovaného textu je potom daný vzťahom $g(s(x)) = f(t(y))$. Urobme substitúciu $s(x) = u$ and $t(y) = v$. Potom pre otvorený text a zašifrovaný text platí $(u^{2^m} + u + \alpha)^{2^m-1} + u = (L_\beta(v) + \gamma)^{2^m-1} + Tr(v)$. Keďže $u^{2^m} + u + \alpha$ a $L_\beta(v) + \gamma$ sa v poli $\mathbb{F}_{2^n}$ nerovnajú nule, platí:

$$\begin{aligned}&(u^{2^m} + u + \alpha)^{2^m}(L_\beta(v) + \gamma) + u(u^{2^m} + u + \alpha)(L_\beta(v) + \gamma) + \\ &+ (u^{2^m} + u + \alpha)(L_\beta(v) + \gamma)^{2^m} + Tr(v)(u^{2^m} + u + \alpha)(L_\beta(v) + \gamma) = 0.\end{aligned} \tag{4.9}$$

Nech $Tr(v) = \zeta_y \in \{0, 1\}$. Prvky poľa $\mathbb{F}_{2^n}$ vieme popísať prvkami poľa $\mathbb{F}_2^n$ (t.j. množinou všetkých n-tíc nad $\mathbb{F}_2$ použitím pevnej bázy $B = \{\vartheta_1, \vartheta_2, ..., \vartheta_n\}$ poľa $\mathbb{F}_{2^n}$ nad $\mathbb{F}_2$. Substitúciou $u = s(x)$ a $v = t(y)$, kde $x = (x_1, x_2, ..., x_n)$ a $y = (y_1, y_2, ..., y_n)$ dostávame n nelineárnych polynomických rovníc, ktoré tvoria *verejný kľúč* kryptosystému.
Tieto rovnice majú vo všeobecnosti tvar:

$$\begin{aligned}&\sum a_{ijk}x_ix_jy_k + \sum b_{ij}x_ix_j + \sum (c_{ij} + \zeta_y)x_iy_j + \\ &+ \sum (d_k + \zeta_y)y_k + \sum (e_k + \zeta_y)x_k + f_l = 0,\end{aligned} \tag{4.10}$$

kde $a_{ijk}, b_{ij}, c_k, d_k, e_k \in \mathbb{F}_2$. Sú tretieho stupňa a teda každá z nich obsahuje $\mathcal{O}(n^3)$ termov. Keďže počet rovníc je n, celková veľkosť verejného kľúča je $\mathcal{O}(n^4)$. Je možné v polynomiálnom čase znížiť zložitosť verejného kľúča na $\mathcal{O}(n^3)$ bez zníženia bezpečnosti kryptosystému [49]. Z rovníc vyplýva, že verejný kľúč je lineárny vzhľadom na bity zašifrovaného textu (respektívne vzhľadom na premenné, reprezentujúce bity zašifrovaného textu, t.j. y_i) a je nelineárny vzhľadom na bity otvoreného textu (respektíve vzhľadom na premenné, reprezentujúce bity otvoreného textu, t.j. x_i).

Systém rovníc (4.10) teda tvorí verejný kľúč. *Súkromný kľúč* kryptosystému pozostáva z parametrov α, β, γ a 2 afinných transformácii (s, t).

Algoritmus 8 popisuje šifrovací algoritmus, resp. Algoritmus 9 popisuje dešifrovací algoritmus tak, ako sú popísané v [49].

Ako je vidieť, dešifrovací algoritmus vráti 4 správy (X_1, X_2, X_3, X_4), pričom jedna z nich je pôvodný otvorený text. Legitímny príjemca potom nemá problém určiť, ktorá

Algoritmus 8: Poly-Dragon: Šifrovací algoritmus

Ak chce Bob poslať správu $x = (x_1, x_2, ..., x_n)$ Alici, urobí nasledovné:

1. Bob dosadí do premenných OT $(x_1, x_2, ..., x_n)$ a zvolí $\zeta_y = 0$ vo verejnom kľúči. Dostane tak n lineárnych rovníc s premennými ZT $(y_1, y_2, ..., y_n)$ nad konečným poľom $\mathbb{Z}_2$. Bob vyrieši tieto rovnice (napr. Gaussovou eliminačnou metódou) a získa vektor $y' = (y'_1, y'_2, ..., y'_n)$.

2. Bob dosadí do premenných OT $(x_1, x_2, ..., x_n)$ a zvolí $\zeta_y = 1$ vo verejnom kľúči. Dostane tak n lineárnych rovníc s premennými ZT $(y_1, y_2, ..., y_n)$ nad konečným poľom $\mathbb{Z}_2$. Bob vyrieši tieto rovnice (napr. Gaussovou eliminačnou metódou) a získa vektor $y'' = (y''_1, y''_2, ..., y''_n)$.

3. Pár (y', y'') tvorí výsledný zašifrovaný text.

Algoritmus 9: Poly-Dragon: Dešifrovací algoritmus

Vstup : Zašifrovaný text (y', y'') a súkromné parametre $\alpha, \beta, \gamma, s, t$
Výstup: Správa $(x_1, x_2, ..., x_n)$

1. $v_1 \leftarrow t(y')$ a $v_2 \leftarrow t(y'')$
2. $z_1 \leftarrow L_\beta(v_1) + \gamma$ a $z_2 \leftarrow L_\beta(v_2) + \gamma$
3. $z_3^{'} \leftarrow z_1^{2^m-1}$ a $z'_4 \leftarrow z_2^{2^m-1}$
4. $z_3 \leftarrow z'_3 + Tr(v_1)$ a $z_4 \leftarrow z'_4 + Tr(v_2)$
5. $z_5 \leftarrow z_3^{2^m} + z_3 + \alpha + 1$ a $z_6 \leftarrow z_4^{2^m} + z_4 + \alpha + 1$
6. $z_7 \leftarrow z_5^{2^m-1}$ a $z_8 \leftarrow z_6^{2^m-1}$
7. $X_1 \leftarrow s^{-1}(z_3 + 1)$, $X_2 \leftarrow s^{-1}(z_4 + 1)$, $X_3 \leftarrow s^{-1}(z_3 + z_7 + 1)$, $X_4 \leftarrow s^{-1}(z_4 + z_8 + 1)$
8. Vráť (X_1, X_2, X_3, X_4)

správa to je.

4.2.4 Použitie kryptosystému PolyDragon v generátore MSTg

Ako bolo spomínané v kapitole 4.2.2, generátor MSTg poskytuje veľkú flexibilitu, keďže nie sú známe obmedzenia na voľbu generátora náhodných pokrytí. My sme sa preto rozhodli skombinovať tento generátor s kryptosystémom Poly-Dragon, pričom tento kryptosystém bol použitý na generovanie prvkov náhodných pokrytí.

Túto „našu“ verziu generátora MSTg sme následne otestovali podobne, ako autori testovali pôvodnú verziu - testovali sme štatistické vlastnosti výstupu pomocou testovacej šablóny NIST [45], aby sme zistili, či sú výstupy takéhoto generátora odlíšiteľné od výstupu nedeterministického generátora. Ďalej sme zisťovali, či zohráva počet náhodných pokrytí, alebo ich štruktúra, významnú úlohu v štatistických vlastnostiach generátora.

Podobne, ako autori MSTg, zvolili sme dve grupy $\mathcal{G}_1$ a $\mathcal{G}_2$, pričom išlo o elementárne abelovské 2-grupy, pričom prvkami grúp boli binárne vektory. Grupovou operáciou bolo modulárne sčítanie mod 2, t.j. operácia XOR ($\oplus$) a rády grúp boli $|\mathcal{G}_1| = 2^{257}$ a $|\mathcal{G}_2| = 2^{129}$.

Fakt 3 *Tu je nutné podotknúť, že rády majú nepárne mocniny ako dôsledok toho, že kryptosystém Poly-Dragon pracuje s nepárnym parametrom n - vid'. Vetu 1 a Vetu 2.*

Na generovanie náhodných pokrytí sme teda použili kryptosystém Poly-Dragon - zapojili sme ho do tzv. „counter“ módu: na začiatku sme vygenerovali náhodné číslo (seed) a konštantu, ktorú sme k seedu pripočítali. Výsledok tvoril nový otvorený text. Ten sme zašifrovali kryptosystémom Poly-Dragon, pričom výsledný zašifrovaný text tvoril prvok náhodného pokrytia. Následne sme k otvorenému textu znovu pripočítali konštantu, výsledok tvoril nový otvorený text, ten sme zašifrovali a dostali ďalší zašifrovaný text, t.j. ďalší prvok náhodného pokrytia. Parametre kryptosystému boli zvolené nasledovne:

- Ireducibilné polynómy - zvolené náhodne príslušného stupňa (257, 129).
- α, γ - zvolené náhodne ($Tr(\alpha) = Tr(\gamma) = 1$).
- β - zvolená ako $1 + \theta$, kde θ je koreň príslušného ireducibilného polynómu v nadpoli.
- s, t - obe afinné transformácie sme zvolili ako identické zobrazenia.

Pomocou tejto konštrukcie sme vygenerovali všetky prvky náhodných pokrytí. Tu nastáva problém, keďže nestačí len vygenerovať pevný počet prvkov a zaradiť ich do príslušných podmnožín $\mathcal{A}_i$. Je potrebné, aby tento súbor podmnožín skutočne tvoril

pokrytie, t.j. aby sa každý prvok grupy dal rozložiť podľa vzťahu (4.7). Tento problém má však efektívne riešenie, resp. existuje ohraničenie, koľko prvkov treba vygenerovať, aby daná množina podmnožín tvorila pokrytie grupy s danou pravdepodobnosťou [50].

Následne sme pomocou generátora vygenerovali dostatočné množstvo bitov na vykonanie štatistického testovania pomocou sady NIST. Celkovo sme testovali 4 rôzne konštrukcie generátora MSTg líšiace sa v počte pokrytí a ich kardinalite.

- MSTg/Poly-Dragon s 3 náhodnými pokrytiami, $|\mathcal{A}_i| = 3$, $C = 1$, ver. $\mathcal{A}$
- MSTg/Poly-Dragon s 4 náhodnými pokrytiami, $|\mathcal{A}_i| = 3$, $C = 1$, ver. $\mathcal{A}$
- MSTg/Poly-Dragon s 5 náhodnými pokrytiami, $|\mathcal{A}_i| = 3$, $C = 1$, ver. $\mathcal{A}$
- MSTg/Poly-Dragon s 6 náhodnými pokrytiami, $|\mathcal{A}_i| = 2^8$, $C = 1$, ver. $\mathcal{A}$

Pre každý typ sme vygenerovali 100 rôznych inštancií generátora, t.j. s rôznymi pokrytiami, a vygenerovali sme 100 binárnych postupností o veľkosti 10^7 bitov. Na štatistické vyhodnotenie sme použili program, ktorý implementuje štatistické testovanie NIST a je vyvíjaný na našom ústave. Celkovo sa na jednu postupnosť aplikovalo 200 testov (t.j. výsledkom bolo 200 p-hodnôt), pričom hladina významnosti bola 0.01. Keďže pre každú verziu generátora sme generovali 100 inštancií, celkovo sme dostali pre každú verziu generátora 100 množín p-hodnôt. Výsledky uvádzame v tabuľke 4.5

Každý riadok tabuľky 4.5 predstavuje jednu konštrukciu (verziu) generátora. Symbol $\#c$ označuje počet náhodných pokrytí, $|\mathcal{A}_i|$ označuje veľkosť podmnožiny $|\mathcal{A}_i|$ danej grupy, f_i označuje počet množín p-hodnôt, ktoré obsahujú práve i nevyhovujúcich p-hodnôt. Stĺpec f_{max} udáva maximálny počet nevyhovujúcich p-hodnôt v jednej množine. Napríklad, prvá konštrukcia obsahovala 3 náhodné pokrytia, pričom jedna podmnožina v pokrytí mala 3 prvky; 38 inštancií tejto verzie generátora nebolo odmietnutých ako dobrý generátor náhodných čísel a najväčší počet nevyhovujúcich p-hodnôt v jednej množine p-hodnôt bol osem.

Verzia [b]	Výstup [b]	$\#c$	$\lvert\mathcal{A}_i\rvert$	f_0	f_1	f_2	f_3	f_4	f_{5+}	f_{max}
257/129	129	3	3	38	38	16	6	1	1	8
257/129	129	4	3	42	37	14	2	2	3	7
257/129	129	5	3	41	33	18	5	1	2	6
257/129	129	5	256	47	34	12	4	1	2	6

Tabuľka 4.5: MSTg/Poly-Dragon - štatistické testovanie normou NIST

Z tabuľky vyplýva, že v našom prípade hral počet pokrytí malú úlohu. Vo všetkých troch prípadoch bola úspešnosť generátora na hranici 40%.

Avšak, vidíme, že keď sme zmenili počet prvkov v jednej podmnožine pokrytia (porovnaj 3. a 4. riadok tabuľky 4.5), potom stúpla úspešnosť generátora na 47%.

Vykonali sme ešte jeden test - testovali sme výstupy kryptosystému Poly-Dragon, aby sme videli „náhodnosť“ samotných pokrytí. Vygenerovali sme 100 inštancií kryptosystému a s každým sme vygenerovali postupnosť výstupných bitov o veľkosti 10^7. Výsledky sú v tabuľke 4.6

Verzia [b]	f_0	f_1	f_2	f_3	f_4	f_{5+}	f_{max}
257	20	27	30	13	14	6	8

Tabuľka 4.6: Poly-Dragon - štatistické testovanie normou NIST

Toto je zaujímavý výsledok, keďže len pätina inštancií nebola odmietnutá normou NIST. Je to menší počet, ako bol počet neodmietnutých generátorov, avšak je zaujímavé, že výsledný generátor mal úspešnosť až 47%. Pre porovnanie opakujeme, že úspešnosť pôvodnej verzie generátora MSTg mala úspešnosť 48% až 51%. Uvedené výsledky boli publikované v [30].

4.2.5 Bezpečnosť generátora MSTg/Poly-Dragon

Keďže naša verzia generátora neobsahuje zásadné zmeny v jeho algoritme, očakávame, že sa nezmenila ani jeho bezpečnosť v porovnaní s pôvodnou verziou, predstavenou v článku [35]. Naša zmena zahŕňala len generovanie náhodných pokrytí pomocou výstupu kryptosystému Poly-Dragon namiesto výstupu nejakého generátora náhodných čísel.

Bezpečnosť pôvodného generátora je načrtnutá v pôvodnom článku [35]. Autori tvrdia, že útok hrubou silou na generátor by mal zložitosť $\mathcal{O}(2^{e_1-e_2-\delta-1})$, kde e_1 je bitová dĺžka prvkov prvej grupy $\mathbb{Z}_{2^{e_1}}$, e_2 je bitová dĺžka prvkov druhej grupy $\mathbb{Z}_{2^{e_2}}$ a $0 \leq \delta \leq 2$, a teda ak e_1 a e_2 sú dostatočne veľké, napr. $e_1 - e_2 \geq 100$, potom je výpočtovo zložité určiť počiatočný seed generátora. Ďalšou zaujímavou vlastnosťou generátora je, že pre dva rôzne vstupy x a x' zobrazenia $\breve{\alpha}$ sa výstupy $\breve{\alpha}(x)$ a $\breve{\alpha}(x')$ líšia približne v polovici bitov. Avšak, presný odhad zložitosti výpočtu seedu s pre daný výstup generátora $(z_1, z_2, ..., z_t)$ nie je známy.

Prvky náhodných pokrytí sú výstupy kryptosystému Poly-Dragon zapojeného v CTR móde. Vygenerovali sme náhodný seed, zašifrovali ho a výsledný zašifrovaný text tvoril prvok pokrytia. Potom sme zvýšili seed o fixnú konštantu, zašifrovali ho a výsledok tvoril ďalší prvok pokrytia, atď. Čiže, ak by bol útočník schopný

dešifrovať prvok pokrytia, respektíve dva po sebe idúce prvky, vedel by počítať ďalšie prvky pokrytia. Lenže, bezpečnosť kryptosystému Poly-Dragon je založená na MQ-probléme, ktorý patrí do kategórie NP-úplných problémov, čiže predpokladáme, že pri správnej voľbe parametrov je tento útok odolný voči útoku hrubou silou. Autori kryptosystému tvrdia, že je odolný aj voči algebraickej kryptoanalýze [49].

Tvrdíme, že bezpečnosť našej varianty generátora je porovnateľná s bezpečnosťou pôvodného generátora. Bezpečnosť generovania prvkov pokrytia spočíva v tom, že útočník nie je schopný zlomiť inštanciu kryptosystému Poly-Dragon v reálnom čase (čiže nie je schopný predikovať/počítať ďalšie prvky). Preto veríme, že ak sa nájde útok, ktorý by vedel predikovať ďalšie prvky pokrytia, dal by sa použiť na lámanie tohto kryptosystému. A ak sa nájde útok, ktorý by dokázal určiť pôvodný seed s generátora z výstupu $(z_1, z_2, ..., z_t)$, potom sa s veľkou pravdepodobnosťou dá tento útok použiť na pôvodný generátor MSTg.

4.2.6 Záver

Predstavili sme kombináciu generátora MSTg s kryptosystémom Poly-Dragon. Vykonali sme štatistické testy, z ktorých usudzujeme, že štruktúra pokrytia grupy zohráva v štatistických vlastnostiach väčšiu úlohu ako počet týchto náhodných pokrytí, čím sme potvrdili tvrdenie autorov generátora MSTg.

Kapitola 5

Záver

Táto dizertačná práca je koncipovaná ako komentovaný súbor 4 článkov, ktoré autor tejto dizertačnej práce publikoval počas svojho doktorandského štúdia.

Cieľom práce bolo prispieť k problematike prúdových šifier chybovou analýzou niektorého z finalistov projektu eSTREAM a návrhom novej prúdovej šifry so zaujímavým dizajnom. Prvý bod pokrýva kapitola 3, ktorá sa venuje chybovej analýze 2 prúdových šifier - šifry LILI-128 a šifry Trivium. Šifra Trivium je finalista projektu eSTREAM, resp. súčasť jeho hardvérového portfólia. Z našich experimentov vyplýva, že technikou fázového posunu je možné nájsť vnútorný stav za použitia 2 fázových posunov a 120 bitového prúdového kľúča, t.j. z 360 vygenerovaných bitov, čo je zlepšenie najlepšieho chybového útoku preklopením bitov, ktorý na nájdenie vnútorného stavu používa dve indukcie jedno-bitových chýb a 420 bitov prúdového kľúča, t.j. 1260 vygenerovaných bitov. Ďalej sme v kapitole 3 popísali vylepšenú verziu útoku indukciou jedno-bitových chýb na šifru LILI-128, kde sme znovu použitím fázového posunu dátového registra dosiahli cca. 20-násobné zníženie počtu indukovaných chýb oproti pôvodnému chybovému útoku pri rovnakom počte bitov prúdového kľúča. Taktiež nižší počet indukovaných chýb so sebou nesie nižšie riziko poškodenia zariadenia.

Návrhu nových prúdových šifier sa venuje kapitola 4. V nej sme popísali prúdovú šifru, ktorej dizajn bol inšpirovaný sovietskym šifrátorom Fialka M-125, ktorý je dodnes považovaný za neprelomený. Hľadali sme kompromis medzi náhodnosťou výstupu šifrátora a jeho nastavením (počet kôl, použité operácie). Zistili sme, že verzia našej prúdovej šifry s 8-bitovým blokom, 6 kolami a modulárnym sčítaním a odčítaním, resp. xorovaním, dosahujúca rýchlosť 208 megabitov za sekundu, prechádza štatistickými testami na hladine významnosti $\alpha = 0.01$. Druhým návrhom prezentovaným v kapitole 4 je kombinácia generátora MSTg a kryptosystému Poly-Dragon, v ktorom sme na generovanie pokrytia grupy použili kryptosystém Poly-Dragon. Táto konštrukcia má porovnateľné štatistické vlastnosti s pôvodným generátorom MSTg. Navyše bezpečnosť použitých náhodných pokrytí, t.j. možnosť predpovedať ďalšie prvky pokrytia zo znalosti po sebe idúcich prvkov, je založená na post-kvantovom šifrovacom algoritme Poly-Dragon.

Literatúra

[1] ADAMKO, L. - JÓKAY, M. - VOJVODA, M. Statistical Analysis of ECRYPT eSTREAM Phase3 Ciphers. In *EE časopis pre elektrotechniku a energetiku*, 2008. p. 193-196.

[2] ANDERSON, R. Optical Fault Induction Attacks. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, 2002, s. 2 - 12.

[3] ANTAL, E. - HROMADA, V. A New Stream Cipher Based on Fialka M-125. In *Tatra Mountains Mathematical Publications*, Bratislava. 2013, s. 101 - 118.

[4] ANTAL, E. - ZAJAC, P. Analysis of the Fialka M-125 Cipher-machine. To appear in *Cryptologia*.

[5] BAO, F. et al. Breaking Public-key Cryptosystems on Tamper Resistant Devices in the Presence of Transient Faults. In *Proceedings of the 5th Workshop on Secure Protocols, LNCS 1361*, Springer-Verlag, 1997, s. 115 - 124.

[6] BARD, G. *Algebraic Cryptanalysis.* Springer, 2009.

[7] BIHAM, E., SHAMIR, A. *A New Cryptanalytic Attack on DES: Differential Fault Analysis*, 1996.

[8] BLUM, L. - BLUM, M. - SHUB, M. A Simple Unpredictable Pseudo-Random Number Generator. In *Proceedings of SIAM J. Comput.*, 1986, s. 364 - 383.

[9] BONEH, D. - DEMILLO, R. - LIPTON, R. On the Importance of Checking Cryptographic Protocols for Faults. In *Journal of Cryptology*, Springer-Verlag, 2001, s. 101 - 119.

[10] COURTOIS, N., et al. ElimLin Algorithm Revisited. In *Fast Software Encryption.* Springer Berlin Heidelberg, 2012, p. 306-325.

[11] COURTOIS, N., et al. Efficient Algorithms for Solving Overdefined Systems of Multivariate Polynomial Equations. In *Advances in Cryptology—EUROCRYPT 2000.* Springer Berlin Heidelberg, 2000. p. 392-407.

[12] COURTOIS, N. - BARD, G. - WAGNER, D. Algebraic and Slide Attacks on KeeLoq. In *Fast Software Encryption.* Springer Berlin Heidelberg, 2008, p. 97-115.

[13] COURTOIS, N. - MEIER, W. Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Proceedings of the 22nd international conference on Theory and applications of cryptographic techniques*, Eurocrypt, 2003, s. 345 - 359.

[14] DAWSON, E. et al. *The LILI-128 Keystream Generator*, 2000.

[15] DE CANNIERE, C. Trivium: A Stream Cipher Construction Inspired by Block Cipher Design Principles. In *Information Security.* Springer Berlin Heidelberg, 2006, p. 171-186.

[16] EIBACH, T. - PILZ, E. - VÖLKEL, G. Attacking Bivium Using SAT Solvers. In *Theory and Applications of Satisfiability Testing–SAT 2008.* Springer Berlin Heidelberg, 2008, p. 63-76.

[17] ENGELS, D., et al. Hummingbird: Ultra-lightweight Cryptography for Resource-constrained Devices. In *Financial Cryptography and Data Security.* Springer Berlin Heidelberg, 2010. p. 3-18.

[18] GOLIC, J. - MORGARI, G. On the Resynchronization Attack. In *Proceedings of FSE*, 2003, s. 100 - 110.

[19] GOMUŁKIEWICZ, Marcin, et al. Synchronization Fault Cryptanalysis for Breaking A5/1. In *Experimental and Efficient Algorithms.* Springer Berlin Heidelberg, 2005, p. 415-427.

[20] GROŠEK, O. - VOJVODA, M. - ZAJAC, P. *Klasické šifry.* STU, 2007.

[21] HAGAI, B. et al. The Sorcerers' Apprentice Guide to Fault Attacks. In *Proceedings of The IEEE*, 2004, č. 94, s. 370 - 382.

[22] HOCH, J. *Fault Analysis of Stream Ciphers* : diplomová práca. Rehovot:Weizmann Institute of Science. 2004. 48 s.

[23] HOCH, J. - SHAMIR, A. Fault Analysis of Stream Ciphers. In *Chryptographic Hardware and Embedded Systems – CHES 2004, Lecture Notes in Computer Science*, Springer-Verlag, 2004, s. 240 - 253.

[24] HOJSIK, M. - RUDOLF, B. Differential Fault Analysis of Trivium. In *Fast Software Encryption*, Springer Berlin Heidelberg, 2008, p. 158-172.

[25] HOJSIK, M. - RUDOLF, B. Floating Fault Analysis of Trivium. In *Progress in Cryptology-INDOCRYPT 2008*, Springer Berlin Heidelberg, 2008, p. 239-250.

[26] HROMADA, V. *Chybová analýza prúdových šifier* : diplomová práca. FEI STU. 2011. 50 s.

[27] HROMADA, V. - VARGA, J. Phase-shift Fault Analysis of Trivium. *Under review in Studia Scientiarum Mathematicarum Hungarica*, 2014.

[28] HROMADA, V. - VOJVODA, M. Fault Analysis of Stream Ciphers. In *Mikulášská kryptobesídka 2011 : Sborník příspěvků: Praha, 1. - 2. decembra 2011*. Praha: Trusted Network Solutions, 2011, s. 69 - 70.

[29] HROMADA, V. - VOJVODA, M. A Note on Poly-Dragon Cryptosystem. In *ELITECH'12 : 14th Conference of Doctoral Students. Bratislava, Slovak Republic, 22 May 2012*, Bratislava: Nakladateľstvo STU, 2012, s. 5.

[30] HROMADA, V. - VOJVODA, M. Using Public-Key Cryptosystem Poly Dragon to Create a PRNG Based on Random Covers for Finite Groups. In *ISCAMI 2012 : Book of abstracts. Malenovice, Czech Republic, 10. - 13.5.2012*, Ostrava: University of Ostrava, 2012, s. 44.

[31] HROMADA, V. - VOJVODA, M. Using Poly-Dragon Cryptosystem in a Pseudorandom Number Generator MSTg. *Accepted for publication in Tatra Mountains Mathematical Publications*.

[32] KOCHER, P. Timing Attacks on Implementation of Diffie-Hellman, RSA, DSS and Other Systems. In *Advances in Cryptology: Proceedings of CRYPTO '96*, Springer-Verlag, 1996, s. 104 - 113.

[33] KOCHER, P. - JAFFE, J. - JUN, B. Differential Power Analysis. In *Lecture Notes in Computer Science*, 1999, s. 388 – 397.

[34] LOE, C. W. - KHOO, K. Side Channel Attacks on Irregularly Decimated Generators. In *Information Security and Cryptology - ICISC 2007*, Springer Berlin Heidelberg, 2007, p. 116-130.

[35] MARQUARDT, P. - SVABA, P. - van TRUNG, T. Pseudorandom Number Generators Based on Random Covers for Finite Groups. In *Designs, Codes and Cryptography*, 2011, s. 1 - 12.

[36] MCDONALD, C., et al. An Algebraic Analysis of Trivium Ciphers Based on the Boolean Satisfiability Problem. *IACR Cryptology ePrint Archive*, 2007, 2007:129.

[37] MEIER, W. - STAFFELBACH, O. Fast Correlation Attacks on Certain Stream Ciphers. In *Journal of Cryptology*, 1989, s. 159 - 176.

[38] MENEZES, A. - OORSCHOT, P. - VANSTONE, S. *Handbook of Applied Cryptography*. 1.vyd. CRC Press, 1994.

[39] MOHAMED, S. E. M., et. al. Using Sat Solving to Improve Differential Fault Analysis of Trivium. In *Information Security and Assurance*, Springer Berlin Heidelberg, 2011, p. 62-71.

[40] PATARIN, J. Asymetric Cryptography with a Hidden Monomial. In *Advances in Cryptology - CRYPTO '96*, Springer-Verlag, 1996, s. 45 - 60.

[41] PERERA, T. - HAMER, D. *General Introduction: Russian Cold War Era M-125 and M-125-3MN Fialka Cipher Machines*, 2005.

[42] QUISQUATER, J. J. *Side-channel Attacks*, 2002.

[43] REUVERS, P. - SIMONS, M. *Fialka M-125: Detailed Description of the Russian Fialka Cipher Machines*. PAHJ Reuvers & MJH Simons, 2009.

[44] ROBSHAW, M. - BILLET, O. *New Stream Cipher Designs: the eSTREAM Finalists.* Springer, 2008.

[45] RUKHIN, A., et. al. *Statistical test suite for random and pseudorandom number generators for cryptographic applications*. Dostupné na internete: `http://csrc.nist.gov/rng`.

[46] SARKAR, P. - MAITRA, S. Nonlinearity Bounds and Constructions of Resilient Boolean Functions. In *LNCS 1880*, Springer-Verlag, 2000, s. 515 - 532.

[47] SHOUP, V. *Number Theory Library*. Dostupné na internete: `http://www.shoup.net/ntl/`.

[48] SIMONETTI, I. - FAUGERE, J. - PERRET, L. Algebraic Attack Against Trivium. In *First International Conference on Symbolic Computation and Cryptography*, SCC, 2008, p. 95-102.

[49] SINGH, R.P. - SAIKIA, A. - SARMA, B.K. Poly-Dragon: an efficient multivariate public key cryptosystem. In *Journal of Mathematical Cryptology*, 2010, s. 365 - 373.

[50] SVABA, P. - van TRUNG, T. On Generation of Random Covers for Finite Groups. In *Tatra Mountains Mt. Math. Publ*, 2007, s. 105 - 112.

[51] YUPU, H. et. al. Fault Analysis of Trivium. In *Designs, Codes and Cryptography.* 2012, s. 289 - 311.

[52] ZAJAC, P. Solving Trivium-based Boolean Equations Using the Method of Syllogisms. *Fundamenta Informaticae*, 2012, p. 359-373.

[53] ZAJAC, P. A New Method to Solve MRHS Equation Systems and Its Connection to Group Factorization. *Journal of Mathematical Cryptology*, 2013, p. 367-381.

[54] ZAJAC, P. - JÓKAY, M. On S-boxes with Low Multiplicative Complexity. In *Tatracrypt 2012: 12th Central European Conference on Cryptology. Smolenice, Slovak Republic*. Slovak Academy of Sciences, 2012, p. 47-48.

Dodatok A

Softvérová implementácia

V tejto prílohe uvádzame referenčnú implementáciu 8-bitovej verzie prúdovej šifry z časti 4.1 v jazyku C. Príslušné S-boxy musia byť vložené do zdrojového kódu.

```
1  #ifndef _FIALKACIPHER_H
2  #define _FIALKACIPHER_H
3
4  #include <stdint.h> // uint64_t
5  #include <math.h>
6
7  typedef unsigned char byte;
8  typedef uint64_t pin_t;
9
10 #define ROT     10   // number of rotoros
11 #define N       8    // N-bit cipher
12 #define MOD     256  // pow(2,n)
13
14 #define XOR 1 // comment to use modular operation
15
16 const byte REF_E[MOD]       = {/*Reflector S-box*/};
17 const byte REF_D[MOD]       = {/*inverse ref. S-box*/};
18 const byte SBOX[MOD]        = {/*rotor S-box*/};
19 const byte SBOX_INV[MOD]    = {/*inverse rotor S-box*/};
20
21 pin_t pins[ROT][4]; // 1 binary vector - 4 * 64 bits
22 byte lambda[ROT];   // 1 lambda - 8 bits
23
24 void setKey(pin_t k1[][4], byte k2[]){
25     for(int i=0; i < ROT; i++){
26         for (int j=0; j < 4; j++) {
27             pins[i][j] = k1[i][j];
28         }
29         lambda[i] = k2[i];
```

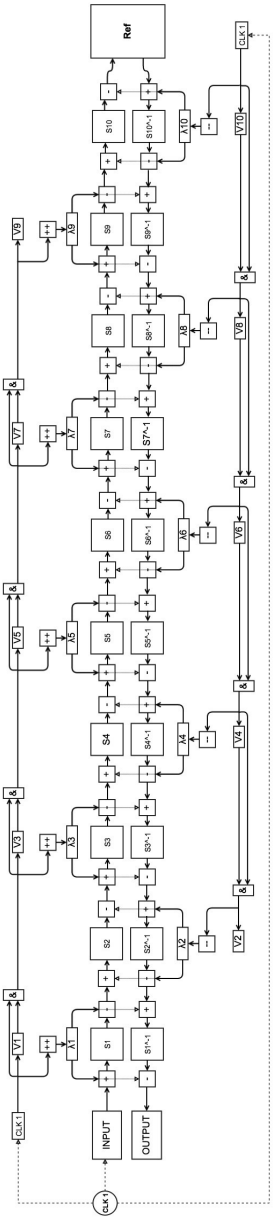
```
30      }
31  }
32
33  /* function & method prototypes */
34   void wheelStepping(void);
35   byte rotors(byte input);
36   byte rotorsInv(byte input);
37
38  void encrypt(byte PT[],byte CT[],const long size);
39  void decrypt(byte PT[],byte CT[],const long size);
40
41  /* impl.*/
42  inline void wheelStepping(void){
43      for(register short i=1; i < 10; i=i+2){
44          pins[i][0] = ((pins[i][0]>>1)|(pins[i][1]<<63));
45          pins[i][1] = ((pins[i][1]>>1)|(pins[i][2]<<63));
46          pins[i][2] = ((pins[i][2]>>1)|(pins[i][3]<<63));
47          pins[i][3] = ((pins[i][3]>>1)|(pins[i][0]<<63));
48          lambda[i]++;
49          lambda[i]%=MOD;
50          if((pins[i][0]&0x0000000000000001) == 0) break;
51      }
52      for(register short i=8; i >= 0; i=i-2){
53          pins[i][0] = ((pins[i][0]<<1)|(pins[i][3]>>63));
54          pins[i][1] = ((pins[i][1]<<1)|(pins[i][0]>>63));
55          pins[i][2] = ((pins[i][2]<<1)|(pins[i][1]>>63));
56          pins[i][3] = ((pins[i][3]<<1)|(pins[i][2]>>63));
57          lambda[i] += MOD -1;
58          lambda[i]%=MOD;
59          if((pins[i][0]&0x0000000000000001) == 0) break;
60      }
61
62  }
63
64  inline byte rotors(const byte input){
65      register byte x = input;
66  #ifdef XOR /* XOR */
67      for(register short i=0; i < 10; i++){
68          x = x^lambda[i];
69          x = SBOX[x];
70          x = x^lambda[i];
71      }
72  #else /* +,- MOD 2^n */
73      for(register short i=0; i < 10; i++){
```

```
74          x+= lambda[i], x %= MOD;
75          x = SBOX[x];
76          x += MOD - lambda[i], x %= MOD;
77      }
78  #endif
79      return x;
80  }
81
82  inline byte rotorsInv(byte input){
83      register byte x = input;
84  #ifdef XOR /* XOR */
85      for(register short i=9; i >= 0; i--){
86          x = x^lambda[i];
87          x = SBOX_INV[x];
88          x = x^lambda[i];
89      }
90  #else /* +,- MOD 2^n */
91      for(register short i=9; i >= 0; i--){
92          x+= lambda[i], x %= MOD;
93          x = SBOX_INV[x];
94          x += MOD - lambda[i], x %= MOD;
95      }
96  #endif
97      return x;
98  }
99
100 inline void encrypt(byte *PT,byte *CT, const long size){
101     for(long i=0; i < size; i++){
102         byte x = PT[i];
103         x = rotors(x);
104         x = REF_E[x];
105         x = rotorsInv(x);
106         wheelStepping();
107         CT[i] = x;
108     }
109 }
110
111 inline void decrypt(byte *PT, byte *CT, const long size){
112   for(long i=0; i < size; i++){
113         byte x = CT[i];
114         x = rotors(x);
115         x = REF_D[x];
116         x = rotorsInv(x);
117         wheelStepping();
```

```
118             PT[i] = x;
119         }
120     }
121 #endif
```

Dodatok B

Dizajn šifry



Obr. B.1: Návrh novej šifry z kapitoly 4.1