FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Algebraická kryptoanalýza zjednodušených variant šifry Grain-128AEAD
Student:	Bc. Daniel Minarovič
Vedoucí:	Mgr. Martin Jureček, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	do konce letního semestru 2023/2024

Pokyny pro vypracování

Proudová šifra Grain-128AEAD byla vybrána do finálového kola v soutěži Lightweight Cryptography, kterou organizuje společnost NIST. V současné době existuje druhá verze šifry, Grain-128AEADv2, pro kterou probíhá intenzivní výzkum zaměřující se na její prolomení. Tato práce se zabývá algebraickou kryptoanalýzou, která danou šifru převede na soustavu rovnic a následně ji vyřeší. Aby výpočet soustavy netrval příliš dlouho, algebraická kryptoanalýza bude aplikována pouze na zjednodušené varianty šifry Grain-128AEADv2.

Podrobné pokyny:
1) Navrhněte škálovatelnou zjednodušenou variantu šifry Grain-128AEADv2.
2) Naimplementujte převod šifry z kroku 1) do soustavy polynomiálních rovnic.
3) Stručně popište základy teorie Groebnerových bází a aplikujte ji při řešení soustav polynomiálních rovnic.
4) Použijte vhodný program (např. Magma) pro výpočet Groebnerových bází a proveďte diskusi o výsledcích.

Elektronicky schválil/a prof. Ing. Róbert Lórencz, CSc. dne 22. prosince 2022 v Praze.

Diplomová práca

ALGEBRAICKÁ KRYPTOANALÝZA ZJEDNODUŠENÝCH VARIANT ŠIFRY GRAIN-128AEAD

Bc. Daniel Minarovič

Fakulta informačních technologií
Katedra informační bezpečnosti
Vedúci: Mgr. Martin Jureček, Ph.D.
3. mája 2023

Obsah

Zoznam obrázkov

Zoznam tabuliek

Chcel by som poďakovať svojmu vedúcemu záverečnej práce, Mgr. Martinovi Jurečkovi, Ph.D. za jeho odborné vedenie, cenné rady a čas, ktorý mi poskytol počas konzultácií.

Vyhlásenie

Prehlasujem, že som predloženú prácu vypracoval(a) samostatne a že som uviedol(uviedla) všetky informačné zdroje v súlade s Metodickým pokynom o etickej príprave vysokoškolských záverečných prác.

Beriem na vedomie, že sa na moju prácu vzťahujú práva a povinnosti vyplývajúce zo zákona č. 121/2000 Sb, autorského zákona, v znení neskorších predpisov. V súlade s ustanovením § 46 odst. 6 tohoto zákona týmto udeľujem bezvýhradné oprávnenie (licenciu) k používaniu tejto mojej práce, a to vrátane všetkých počítačových programov ktoré sú jej súčasťou alebo prílohou a tiež všetkej ich dokumentácie (ďalej len „Dielo“), a to všetkým osobám, ktoré si prajú Dielo používať. Tieto osoby sú oprávnené Dielo používať akýmkoľvek spôsobom, ktorý neznižuje hodnotu Diela, ale len pre nezárobkové účely. Toto oprávnenie je časovo, územne a množstevne neobmedzené.

V Praze dne 3. mája 2023 .....

Abstrakt

Táto práca sa zaoberá algebraickou kryptoanalýzou, ktorá šifru Grain-128AEADv2, ďalej už len Grain, prevedie na sústavu polynomiálnych rovníc a následne ju vyrieši pomocou Groebnerových báz. Algebraická kryptoanalýza bude aplikovaná na zjednodušené varianty šifry Grain. Pre efektívnejší výpočet riešenia použijeme metódu Locality Sensitive Hashing pre zjednodušenie systému polynomiálnych rovníc. V rámci experimentov sme boli schopný prelomiť napríklad zmenšenú 16 bitovú variantu šifry s počtom kôl 64 a taktiež aj 24 bitovú variantu s počtom kôl 8. Čas generovania rovníc pre zmienenú 24 bitovú variantu trval približne 8 minút a výpočet riešenia 113 sekúnd. Taktiež sme aplikovali algoritmus Locality Sensitive Hashing a tým sme zrýchlili výpočet Groebnerových báz a riešenia. Napríklad pre 24 bitovú verziu s počtom kôl 8 sme boli schopný znížiť výpočet riešenia zo 113 sekúnd na 1,88 sekundy.

Klíčová slova algebraická kryptoanalýza, Groebnerovy bázy, Grain-128AEADv2, LSH

Abstract

This master thesis deals with algebraic cryptanalysis which converts the Grain-128AEADv2 cipher, hereinafter referred to as Grain, into a system of polynomial equations and then solves it using Groebner bases. Algebraic cryptanalysis will be applied to small scale variants of Grain cipher. For a more efficient calculation of the solution, we will use the Locality Sensitive Hashing method for simplifying the system of polynomial equations. As part of the experiments, we were able to break, for example, a 16-bit small scale variant of the cipher with the number of rounds of 64, as well as the 24-bit variant with number of rounds of 8. The generation time of the equations for the mentioned 24-bit variant took approximately 8 minutes, and the calculation of the solution took 113 seconds. We also applied the Locality Sensitive Hashing algorithm and thereby accelerated the calculation of Groebner bases and solutions. For example, for the 24-bit version with the number of rounds of 8, we were able to reduce the calculation of the solution from 113 seconds to 1,88 seconds.

Keywords algebraic cryptanalysis, Groebner bases, Grain-128AEADv2, LSH

Zoznam skratiek

NFSR	Non-linear Feedback Shift Register
LFSR	Linear Feedback Shift Register
MAC	Message Authentication Code
NIST	National Institute of Standards and Technology
AEAD	Authenticated Encryption with Associated Data
SSC	Small-state stream ciphers
DER	Distinguished Encoding Rules
LSH	Locality Sensitive Hashing

Úvod

Táto diplomová práca sa bude zaoberať kryptoanalýzou šifry Grain-128-AEADv2 [1], ktorá patrí medzi hardvérovo efektívne synchrónne prúdové šifry. Jej predchodca Grain-128AEAD bol vybraný do finálového kola súťaže Lightweight Cryptography, ktorú organizuje Národný inštitút pre štandardy a technológie. Jedná sa o súťaž pre algoritmy, vhodné na použitie v obmedzených podmienkach.

V princípe ide o autentizačný šifrovací algoritmus s podporou pre asociované dáta. Špecifikácia daného algoritmu je úzko spätá so šifrou Grain-128a [2], ktorá bola zavedená v roku 2011 a stále odoláva rôznym analýzam. Šifrovací algoritmus v tejto práci je navrhnutý tak, aby mal čo najmenej zmien oproti vyššie spomínanému Grain-128a. Oproti tomu Grain-128a je založený na dôkladne analyzovaných šifrách jeho predchodcov, ktoré poskytujú dôležité informácie o bezpečnosti dizajnu. Vďaka tomu sa autori môžu opierať o bezpečnosť už doteraz neprelomeného Grain-128a.

Jednou z výraznejších zmien, ktoré boli prevedené, je zvýšená bezpečnosť proti rekonštrukcii kľúča zo známeho vnútorného stavu. To by mala byť motivácia pre aktualizáciu na túto novú verziu AEADv2. Ako už bolo zmienené, jedná sa o prúdovú šifru, ktorej vstupom je otvorený text s premenlivou dĺžkou, asociované dáta s premenlivou dĺžkou, nonce s pevnou dĺžkou o veľkosti 96 bitov a kľúč s pevnou dĺžkou o veľkosti 128 bitov. Výstupom je šifrovaný text s premenlivou dĺžkou. Otvorený text sa dá získať len z platného šifrovaného textu.

V práci podrobíme šifru Grain-128-AEADv2 skúške odolnosti voči algebraickej kryptoanalýze [3]. Princíp algebraickej kryptoanalýzy je založený na prevode problému prelomenia kryptosystému na problém riešenia sústavy polynomiálnych rovníc nad konečným poľom. Postup sa všeobecne rozdeľuje do dvoch krokov. Zo špecifických vlastností šifry sa odvodí sústava polynomiálnych rovníc nad konečným poľom a následne sa na túto sústavu aplikuje vybraný postup pre výpočet riešenia sústavy. V práci budeme využívať pre riešenie takejto sústavy, práve Groebnerove bázy. S ich pomocou môžeme nájsť sústavu, ktorej riešenie je jednoduchšie ako riešenie sústavy pôvodnej.

Prvá kapitola obsahuje potrebné definície a pojmy pre matematický základ diplomovej práce. V nej si popíšeme práve Groebnerove bázy a algoritmy pre ich výpočet.

Druhá kapitola sa venuje teoretickému základu algoritmu vybranej šifry. Vysvetľuje jednotlivé stavebné bloky, funkcie šifry, inicializáciu a algoritmus pre šifrovanie a dešifrovanie.

V tretej kapitole sú popísané niektoré práce, ktoré sa venujú podobnej téme ako je naša práca.

V kapitole 4 je pozornosť venovaná dokumentácii implementácie jednotlivých skriptov a ich fungovaniu. Kapitola napríklad obsahuje popis algoritmu potrebného k prevodu šifry na polynomiálne rovnice a výpočet riešenia rovníc. V poslednej kapitole sú prezentované výsledky experimentov vykonaných v rámci našej implementácie pre rôzne zmenšené varianty šifry a pokusy na použitie algoritmu pre redukciu vygenerovanej sústavy polynómov.

Kapitola 1
Matematický základ

V tejto kapitole sa pozrieme na všetky matematické pojmy použité v tejto práci. Budeme sa venovať hlavne oblasti týkajúcej sa Groebnerových báz. Z [4] budú prevzaté jednotlivé nižšie uvedené definície a taktiež tu môžeme nájsť dôkazy k uvedeným vetám.

1.1 Polynómy a ideály

Zadefinujeme si základné pojmy, ktoré budeme ďalej v celej práci používať.

▶ **Definícia 1.1.** *Pojmom* ***monóm*** *označujeme súčin v tvare*

$$x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdot \ldots \cdot x_n^{\alpha_n},$$

kde exponenty $\alpha_1, \ldots, \alpha_n$ *sú nezáporné celé čísla. Za celkový stupeň monómu potom budeme považovať súčet* $\mid \alpha \mid = \alpha_1 + \ldots + \alpha_n$.

▶ **Definícia 1.2.** ***Polynóm f*** *s premennými* $x_1, \ldots, x_n$ *nad poľom k je konečná lineárna kombinácia monómov s koeficientami z k. Polynóm f zapisujeme ako*

$$f = \sum_{\alpha} a_\alpha x^\alpha, \quad a_\alpha \in k.$$

V sume vyššie sčítavame cez konečný počet n-tic $\alpha = (\alpha_1, \ldots, \alpha_n)$. *Množinu všetkých polynómov s premennými* $x_1, \ldots, x_n$ *nad poľom k označujeme* $k[x_1, \ldots, x_n]$. *Množinu* $k[x_1, \ldots, x_n]$ *nazývame* ***polynomiálny okruh****.*

▶ Poznámka 1.3. Označením x^α myslíme $x_i^{\alpha_i}$, $1 \leq i \leq n$.

▶ **Definícia 1.4.** *Nech* $f = \sum_\alpha a_\alpha x^\alpha$ *je polynóm z* $k[x_1, \ldots, x_n]$.

(i) a_α *nazývame* ***koeficient*** *monómu* x^α.

(ii) Pokiaľ $a_\alpha \neq 0$, *potom* $a_\alpha x^\alpha$ *označujeme ako* ***term*** *polynómu f.*

(iii) ***Celkový stupeň*** *polynómu* $f \neq 0$, *budeme označovať deg(f), je najväčšie* $\mid \alpha \mid$ *také, že koeficient* a_α *je nenulový, Celkový stupeň nulového polynómu nie je definovaný.*

▶ **Definícia 1.5.** *Majme prvok* $a \in GF(2)$. *Potom najmenšie* $n > 0$ *také, že* $a^n = e$ *sa nazýva* ***rád prvku*** *a. Ak* $\forall n \in N : a^n \neq e$, *potom hovoríme, že a má rád prvku nekonečno [5].*

▶ **Definícia 1.6.** *(****Primitívny polynóm****) Polynóm* $f(x)$ *stupňa n nad konečným poľom GF(2) je* ***primitívny*** *ak* $f(0) \neq 0$ *a rád prvku, respektíve polynómu je ord* $f(x) = 2^n - 1$ *[6].*

▶ **Tvdenie 1.7.** *Pre akékoľvek prvočíslo q (alebo jeho mocninu) a pre akékoľvek kladné celé číslo n existuje primitívny polynóm stupňa n nad GF(q).*

▶ **Definícia 1.8.** *Majme podmnožinu $I \subseteq k[x_1, \ldots, x_n]$. I je **ideál**, pokiaľ*

(i) $0 \in I$,

(ii) $\forall f, g \in I : f + g \in I$,

(iii) $\forall f \in I, h \in k[x_1, \ldots, x_n] : h \cdot f \in I$.

▶ **Definícia 1.9.** *Majme polynómy $f_1, \ldots, f_s$ z $k[x_1, \ldots, x_n]$. Položme potom*

$$\langle f_1, \ldots, f_s \rangle = \left\{ \sum_{i=1}^{s} h_i f_i \mid h_1, \ldots, h_n \in k[x_1, \ldots, x_n] \right\}$$

▶ **Tvdenie 1.10.** *Pokiaľ máme polynómy $f_1, \ldots, f_s \in k[x_1, \ldots, x_n]$, potom $\langle f_1, \ldots, f_s \rangle$ je ideálom $k[x_1, \ldots, x_n]$. Tento ideál nazývame **ideál generovaný** $f_1, \ldots, f_s$.*

▶ **Definícia 1.11.** *Majme ideál $I \subseteq k[x_1, \ldots, x_n]$. Ak existuje množina $A \subseteq N_0^n$ taká, že I pozostáva zo všetkých polynómov, ktoré sú v tvare*

$$\sum_{\alpha \in A} h_\alpha x^\alpha, \text{ kde } h_\alpha \in k[x_1, \ldots, x_n],$$

*tak potom I nazývame **monomiálny ideál**. Označujeme ho $I = \langle x^\alpha \mid \alpha \in A \rangle$.*

▶ **Definícia 1.12.** *(**Booleovsky polynomiálny okruh**) Komutatívny polynomiálny okruh B s identitou 1 nazývame booleovský ak pre každý prvok a z B platí, že $a^2 = a$ [7].*

1.2 Monomiálne usporiadanie

Pre lepšie pochopenie Groebnerových báz si zadefinujeme rôzne pojmy pre monomiálne usporiadania.

▶ **Definícia 1.13.** *Relácia $>$ na N_0^n je **lineárne usporiadanie**, pokiaľ pre každú dvojicu $\alpha, \beta \in N_0^n$ platí práve jedno z týchto troch tvrdení*

$$\alpha > \beta, \alpha = \beta, \alpha < \beta.$$

▶ **Definícia 1.14.** *Relácia $>$ na N_0^n je **dobré usporiadanie**, pokiaľ pre každú neprázdnu podmnožinu N_0^n platí, že má najmenší prvok vzhľadom k $>$. To znamená, že pre každú neprázdnu podmnožinu $A \subseteq N_0^n$ existuje $\alpha \in A$ také, že $\beta > \alpha$ a pre každé $\beta \in A, \beta \neq \alpha$.*

▶ **Definícia 1.15.** ***Monomiálne usporiadanie** $>$ na $k[x_1, \ldots, x_n]$ je relácia na N_0^n, ktorá splňuje*

(i) $>$ *je lineárne usporiadanie na* N_0^n.

(ii) Pokiaľ $\alpha > \beta$ *a* $\gamma \in N_0^n$, *potom* $\alpha + \gamma > \beta + \gamma$.

(iii) $>$ *je dobré usporiadanie na* N_0^n.

▶ **Definícia 1.16.** *(**Lexikografické usporiadanie**) Nech $\alpha = (\alpha_1, \ldots, \alpha_n)$ a $\beta = (\beta_1, \ldots, \beta_n)$ sú z N_0^n. Nech napríklad je $\alpha >_{lex} \beta$, ak prvý nenulový prvok z ľavej strany rozdielu $\alpha - \beta \in N_0^n$ je kladný. Budeme písať $x^\alpha >_{lex} x^\beta$, pokiaľ $\alpha >_{lex} \beta$.*

▶ **Príklad 1.17.** Uvedieme si príklady pre lexikografické usporiadanie

1. $\alpha = (3,4,5) >_{lex} \beta = (1,4,5)$, *pretože* $\alpha - \beta = (2,0,0)$
2. $\alpha = (7,7,8) >_{lex} \beta = (7,5,9)$, *pretože* $\alpha - \beta = (0,2,-1)$
3. $\alpha = (2,3,4) <_{lex} \beta = (4,1,2)$, *pretože* $\alpha - \beta = (-2,2,2)$

▶ **Definícia 1.18.** *(**Odstupňované lexikografické usporiadanie**) Majme* $\alpha, \beta \in N_0^n$. *Povedzme, že* $\alpha >_{grlex} \beta$, *pokiaľ*

$$| \alpha | = \sum_{i=1}^{n} \alpha_i > | \beta | = \sum_{i=1}^{n} \beta_i$$

alebo

$$| \alpha | = | \beta | \text{ a zároveň } \alpha >_{lex} \beta.$$

▶ **Príklad 1.19.** Pozrieme sa na príklady pre odstupňované lexikografické usporiadanie

1. $\alpha = (2,2,2) >_{grlex} \beta = (1,2,0)$, *pretože* $| \alpha | > | \beta |$, *tzn.* $6 > 3$
2. $\alpha = (1,2,1) <_{grlex} \beta = (3,0,1)$, *pretože* $| \alpha | = | \beta |$, *a* $\alpha - \beta = (-2,2,0)$

▶ **Definícia 1.20.** *(**Obrátené odstupňované lexikografické usporiadanie**) Majme* $\alpha, \beta \in N_0^n$. *Povedzme, že* $\alpha >_{grevlex} \beta$, *pokiaľ*

$$| \alpha | = \sum_{i=1}^{n} \alpha_i > | \beta | = \sum_{i=1}^{n} \beta_i$$

alebo

$$| \alpha | = | \beta | \text{ a zprava prvý nenulový prvok rozdielu } \alpha - \beta \in Z^n \text{ je záporný.}$$

▶ **Príklad 1.21.** Teraz si ukážeme príklady na obrátené odstupňované lexikografické usporiadanie

1. $\alpha = (2,2,2) >_{grevlex} \beta = (1,2,0)$, *pretože* $| \alpha | > | \beta |$, *tzn.*$6 > 3$
2. $\alpha = (1,2,1) <_{grevlex} \beta = (3,0,1)$, *pretože* $| \alpha | = | \beta |$, *a* $\alpha - \beta = (-2,2,0)$

▶ **Definícia 1.22.** *Nech* $f = \sum_\alpha a_\alpha x^\alpha$ *je nenulový polynóm z* $k[x_1, \ldots, x_n]$ *a nech* $>$ *je monomiálne usporiadanie.*

(i) ***Multistupeň*** *polynómu f je*

$$multideg(f) = max(\{\alpha \in Z_{\geq 0}^n \mid a_\alpha \neq 0\}),$$

kde maximum berieme vzhľadom k usporiadaniu $>$.

(ii) ***Vedúci koeficient*** *polynómu f je*

$$LC(f) = a_{multideg(f)} \in k.$$

(iii) ***Vedúci monóm*** *polynómu f je*

$$LM(f) = x^{multideg(f)}.$$

(iiii) ***Vedúci term*** *polynómu f je*

$$LT(f) = LC(f) \cdot LM(f).$$

▶ **Príklad 1.23.** Majme polynóm $f = 3x^6y^2z^2 + xyz^3$ a lexikografické usporiadanie $>_{lex}$. Potom máme

- multideg(f) = (6,2,2)
- LC(f) = 3
- LM(f) = $x^6y^2z^2$
- LT(f) = $3x^6y^2z^2$

1.3 Groebnerove bázy

Teraz si už predstavíme pojem Groebnerova báza a algoritmy používané na jej výpočet.

▶ **Definícia 1.24.** *Ideál $I \subseteq k[x_1, \ldots, x_n]$ je **monomiálny ideál** pokiaľ existuje podmnožina $A \subseteq N_0^n$ taká, že I pozostáva zo všetkých polynómov, ktoré sú konečnými súčtami v tvare*

$$\sum_{\alpha} h_\alpha x^\alpha, \text{ kde } h_\alpha \in k[x_1, \ldots, x_n].$$

Budeme zapisovať ako $I = \langle x^\alpha \mid \alpha \in A \rangle$.

▶ **Tvdenie 1.25.** *Nech $I = \langle x^\alpha \mid \alpha \in A \rangle$ je monomiálny ideál. Potom monóm x^β leží v I práve vtedy, keď x^β je deliteľné x^α pre nejaké $\alpha \in A$.*

▶ **Definícia 1.26.** *Nech $I \in k[x_1, \ldots, x_n]$ je ideál rôzny od 0 a majme monomiálne usporiadanie na $k[x_1, \ldots, x_n]$. Potom*

(i) Množinu vedúcich termov nenulových prvkov z I budeme označovať LT(I). Platí, že

$$LT(I) = \{cx^\alpha \mid \exists f \in I \text{ také, že } LT(f) = cx^\alpha\}.$$

(ii) Ideál generovaný prvkami z LT(I) potom budeme značiť $\langle LT(I) \rangle$.

▶ **Definícia 1.27.** *Majme monomiálne usporiadanie na okruhu polynómov $I \subseteq k[x_1, \ldots, x_n]$. Konečnou podmnožinou $G = \{g_1, \ldots, g_t\}$ ideálu $I \subseteq k[x_1, \ldots, x_n]$ rôzneho od $\{0\}$ nazývame Groebnerova báza pokiaľ*

$$\langle LT(g_1), \ldots, LT(g_t) \rangle = \langle LT(I) \rangle.$$

▶ **Tvdenie 1.28.** *Majme nejaké monomiálne usporiadanie. Potom každý ideál $I \in k[x_1, \ldots, x_n]$ rôzny od $\{0\}$ má Groebnerovu bázu. Každá Groebnerova báza ideálu I je báza ideálu I.*

▶ **Veta 1.29.** ***(Delenie v** $k[x_1, \ldots, x_n]$**)** *Majme monomiálne usporiadanie $>$ na N_0^n a nech $F = (f_1, \ldots, f_s)$ je usporiadaná s-tica polynómov z $k[x_1, \ldots, x_n]$. Potom každé $f \in k[x_1, \ldots, x_n]$ sa dá zapísať ako*

$$f = q_1 f_1 + \ldots + q_s f_s + r,$$

kde $q_i, r \in k[x_1, \ldots, x_n]$ a buď $r = 0$ alebo r je lineárna kombinácia monómov, z ktorých žiadny nie je deliteľný žiadnym termom z $LT(f_i), \ldots, LT(f_s)$ Prvok r nazývame zvyšok po delení F a označujeme ako $\overline{f}^F$.

▶ Poznámka 1.30. Algoritmus delenia polynómov v n premenných nebudeme uvádzať. Môžeme ho nájsť v [4]

▶ **Tvdenie 1.31.** *Nech $G = \{g_1, \ldots, g_t\}$ je Groebnerova báza ideálu $I \subset k[x_1, \ldots, x_n]$ a nech $f \in k[x_1, \ldots, x_n]$. Potom $f \in I$ práve vtedy, keď je zvyšok pri delení f množinou G nula.*

▶ **Definícia 1.32.** ***Redukovaná Groebnerova báza*** *polynomiálneho ideálu I je taká Groebnerova báza ideálu I, pre ktorú platí nasledujúce:*

(i) $LC(p) = 1$ *pre všetky* $p \in G$.

(ii) Žiadny monóm z p *neleží v* $\langle LT(G \backslash \{p\})\rangle$ *pre všetky* $p \in G$.

▶ **Definícia 1.33.** *Nech* $f, g \in k[x_1, \ldots, x_n]$ *sú nenulové polynómy.*

(i) Ak $multideg(f) = \alpha$ *a* $multideg(f) = \beta$, *potom nech* $\gamma = (\gamma_1, \ldots, \gamma_n)$, *kde* $\gamma_i = max(\alpha_i, \beta_i)$ *pre všetky* i. *Monóm* x^γ *nazývame najmenší spoločný násobok LM(f) a LM(g) a píšeme ho ako*

$$x^\gamma = lcm(LM(f), LM(g)).$$

(ii) Ako ***S-polynóm*** f *a* g *označujeme polynóm*

$$S(f,g) = \frac{x^\gamma}{LT(f)} \cdot f - \frac{x^\gamma}{LT(g)} \cdot g.$$

▶ **Príklad 1.34.** Majme polynómy $f = 2x^2y^2 + xy^2$ a $g = x^6y + 2y$ a uvažujeme odstupňované lexikografické usporiadanie $>_{grlex}$.

$$LT(f) = 2x^2y^2, LM(f) = x^2y^2, LT(g) = x^6y, LM(g) = x^6y$$

$$x^\gamma = lcm(LM(f), LM(g)) = x^6y^2$$

$$S(f,g) = \frac{x^6y^2}{2x^2y^2} \cdot f - \frac{x^6y^2}{x^6y} \cdot g$$

$$= \frac{x^4}{2} \cdot f - y \cdot g$$

$$= x^6y^2 + \frac{x^5y^2}{2} - x^6y^2 - 2y^3$$

$$= \frac{x^5y^2}{2} - 2y^3$$

▶ **Veta 1.35.** ***(Buchbergerovo kritérium)*** *Majme polynomiálny ideál I. Potom báza $G = \{g_1, \ldots, g_t\}$ ideálu I je jeho Groebnerova báza práve vtedy, keď pre každú dvojicu $i \neq j$, zvyšok po delení $S(g_i, g_j)$ množinou G je nula. Značíme*

$$\overline{S(g_i, g_j)}^G = 0.$$

Toto kritérium sa využíva aj v algoritme pre výpočet Groebnerových báz. Pomocou neho dokážeme overiť, či daná báza ideálu je skutočne Groebnerova báza. Buchbergerov algoritmus bol prvý krát predstavený v [8]. Nižšie uvedený pseudokód je prevzatý z [4] ako aj ostatné definície.

Algoritmus 1 Buchbergerov algoritmus na výpočet GB

Vstup: *množina polynómov* $F = (f_1, \ldots, f_s)$
Výstup: *Groebnerova báza* $G = (g_1, \ldots, g_t)$ *ideálu* $I = \langle f_1, \ldots, f_s \rangle$

$G \leftarrow F$
repeat
  $G' \leftarrow G$
  for `každý pár p,q,` $p \neq q$ `v` G' **do**
    $S \leftarrow \overline{S(p,q)}^{G}$
    if $S \neq 0$ **then**
      $G \leftarrow G \cup \{S\}$
    end if
  end for
until $G' = G$
return G

1.4 Algoritmus F4

Pre získanie správneho výsledku Groebnerovej bázy ideálu nezáleží na poradí výberu párov polynómov [8]. Teraz si predstavíme algoritmus F4 [9]. Hlavným rozdielom od Buchbergerova algoritmu je to ako sa vyberajú dané páry jednotlivých polynómov používaných k spracovaniu. Pseudokód pre F4 algoritmus uvedený nižšie je taktiež prevzatý z [4].

Algoritmus 2 F4

Vstup: *množina polynómov* $F = (f_1, \ldots, f_s)$
Výstup: *Groebnerova báza* $G = (g_1, \ldots, g_t)$ *ideálu* $I = \langle f_1, \ldots, f_s \rangle$

$G \leftarrow F$
$t \leftarrow s$
$B \leftarrow \{\{i,j\} \mid 1 \leq i < j \leq s\}$
while $B \neq \emptyset$ **do**
  Zvol $B' \neq \emptyset$, $B' \subseteq B$
  $B \leftarrow B \backslash B'$
  $L \leftarrow \bigcup_{\{i,j\} \in B'} \left\{ \frac{lcm(LM(f_i), LM(f_j))}{LT(f_i)} \cdot f_i, \frac{lcm(LM(f_i), LM(f_j))}{LT(f_j)} \cdot f_j \right\}$
  $M \leftarrow Spočítaj M(L, G)$
  $N \leftarrow$ redukovaný odstupňovaný tvar matice M
  $N^+ \leftarrow \{n \in \pi(N) \mid LM(n) \notin \langle LM(\pi(M)) \rangle\}$
  for $n \in N^+$ **do**
    $t \leftarrow t + 1$
    $f_t \leftarrow n$
    $G \leftarrow G \cup \{f_t\}$
    $B \leftarrow B \cup \{\{i,t\} \mid 1 \leq i < t\}$
  end for
end while
return G

Teraz sa pozrieme na jednotlivé premenné algoritmu F4.

1.4.1 Množiny F, G a premenná t

F je vstupná množina polynómov tvoriaca bázu ideálu I. Množina G je výstupom algoritmu. Množina G je na začiatku inicializovaná všetkými polynómami z množiny F. Do nej sa pridávajú ďalšie polynómy až kým sa z nej nestane Groebnerova báza ideálu I. Premenná t reprezentuje aktuálny počet polynómov v množine G.

1.4.2 Množiny B a B'

V množine B sú uložené indexy takých polynómov z G, pre ktoré nie je overené, že

$$S(f_i, f_j) \rightarrow_G 0.$$

To platí pre všetky dvojice polynómov z G pri začiatku algoritmu. Množina B' je podmnožinou množiny B, ktorú sa z nej snažíme odstrániť.

1.4.3 Množina L

Táto množina obsahuje prvky v tvare

$$\frac{lcm(LM(f_i), LM(f_j))}{LT(f_i)} \cdot f_i$$

pre každú dvojicu indexov $\{i, j\} \in B'$. Tento podiel nám dá práve S-polynóm $S(f_i, f_j)$ (z definície 1.33).

1.4.4 matice M

Vstupom do funkcie *SpočítajM* sú práve množiny L a G a výstupom je matica M. Pre porozumenie toho čo daná matica reprezentuje potrebujeme nasledujúce definície.

▶ **Definícia 1.36.** *Množinu monómov z polynómu $f \in k[x_1, \ldots, x_n]$ budeme označovať Mon(f). Ak máme množinu polynómov $K \subseteq k[x_1, \ldots, x_n]$, potom píšeme*

$$Mon(K) = \bigcup_{f \in K} Mon(f).$$

Ak máme navyše nejakú množinu $H = \{f_1, \ldots, f_r\} \subseteq k[x_1, \ldots, x_n]$ a monomiálne usporiadanie $>$, potom $Mon(H) = \{m_1, \ldots, m_s\}$.

▶ **Definícia 1.37.** *Majme*

$$X = \begin{pmatrix} m_{i1} \\ \vdots \\ m_{is} \end{pmatrix} \tag{1.1}$$

je usporiadaná s-tica monómov z Mon(H) taká, že

$$m_{i1} > \ldots > m_{is}.$$

Potom M je matica typu $r \times s$:

$$MX = \begin{pmatrix} f_1 \\ \vdots \\ f_r \end{pmatrix}. \tag{1.2}$$

▶ Poznámka 1.38. Každý riadok matice M je rovný jednému polynómu z množiny H a v danom riadku sú koeficienty jednotlivých monómom z množiny Mon(H).

1.4.5 Funkcia SpočítajM

Ako je implementovaná funkcia SpočítajM môžeme vidieť nižšie na pseudokóde 3.

Algoritmus 3 SpočítajM

Vstup: $L, G = (f_1, \ldots, f_t)$
Výstup: M
1: $H \leftarrow L$
2: $done \leftarrow LM(H)$
3: **while** $done \neq Mon(H)$ **do**
4: vyber najväčší $x^\beta \in (Mon(H) \backslash done)$ vzhľadom k $>$
5: $done \leftarrow done \cup \{x^\beta\}$
6: **if** existuje $f_t \in G$ také že $LM(f_l) \mid x^\beta$ **then**
7: zvol ľubovolné $f_l \in G$ také, že $LM(f_l) \mid x^\beta$
8: $H \leftarrow H \cup \left\{ \frac{x^\beta}{LM(f_l)} \right\}$
9: **end if**
10: **end while**
11: $M \leftarrow$ matica koeficientov polynómov množiny H, ktorej stĺpce odpovedajú prvkom Mon(H) usporiadaným podľa $>$
12: **return** M

Matica M je reprezentovaná množinou H. Množina H splňuje nasledujúce podmienky.

(i) $L \subseteq H$,

(ii) Ak x^β je monóm nejakého polynómu $f \in H$ a zároveň existuje $f_l \in G$ také, že $LM(f_l) \mid x^\beta$, potom H vždy obsahuje prvok $x^\alpha f_l$ taký, že $LM(x^\alpha f_l) = x^\beta$.

Množina H sa inicializuje pomocou množiny L. Tým sa hneď splní podmienka (i) 1.4.5. V podmnožine *done* sa ukladajú tie polynómy, ktoré splňujú podmienku (ii)1.4.5.

1.4.6 Matica *N*

Pre vysvetlenie matice N potrebujeme tieto definície prevzaté z [10].

▶ **Definícia 1.39.** *Vedúci prvok matice je prvý nenulový prvok zľava. Matica je v* ***stupňovitom tvare*** *ak splňuje nasledujúce podmienky:*

(i) Každý nulový riadok matice leží pod tými nenulovými.

(ii) Každý vedúci prvok na všetkých riadkoch matice je vždy viac vpravo než všetky vedúce prvky riadkov nad ním.

▶ **Definícia 1.40.** *Matica je v* ***redukovanom stupňovitom tvare*** *ak splňuje nasledujúce podmienky:*

(i) Každý nulový riadok matice leží pod tými nenulovými.

(ii) Každý vedúci prvok na všetkých riadkoch matice je vždy viac vpravo než všetky vedúce prvky riadkov nad ním.

(iii) Každý vedúci prvok je rovný 1.

(iiii) Každý prvok matice ležiaci v stĺpci nad niektorým vedúcim prvkom je rovný 0.

Matica N je v podstate podobná matici M keďže z nej vznikla. Matica N je ktomu ešte v redukovanom stupňovitom tvare podľa definície 1.40.

1.4.7 Množina N^+

Do matice N^+ sa priradia iba tie riadky z matice N, ktorých vedúce termy nie sú deliteľné vedúcimi termami žiadneho polynómu z matice M.

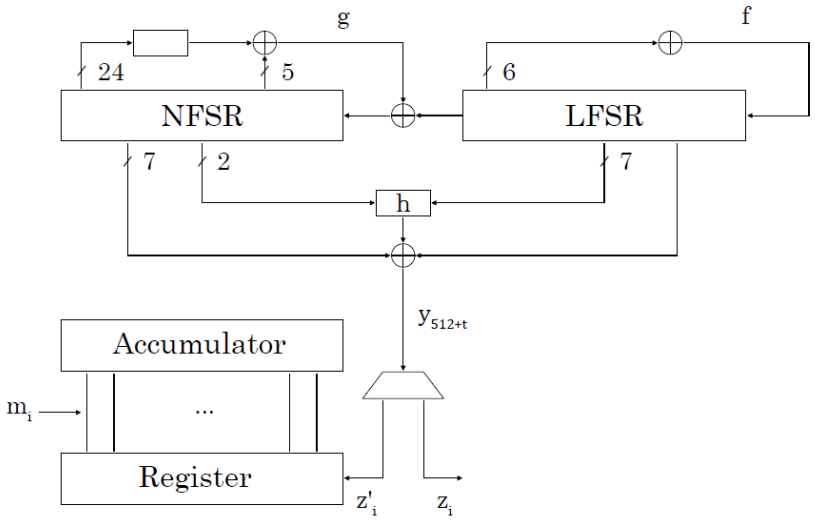
Kapitola 2

Teoretický základ algoritmu

V danej kapitole sme vychádzali zo špecifikácie Grain-128AEADv2 [1]. Jeho základom sú dva hlavné stavebné bloky. Prvým z nich je generátor predvýstupu, ktorého zloženie pozostáva z posuvného registra s lineárnou spätnou väzbou (anglicky Linear Feedback Shift Register, skrátene LFSR), posuvného registra s nelineárnou spätnou väzbou (anglicky Non-linear Feedback Shift Register, skrátene NFSR) a funkcie predvýstupu. Druhý je autentifikačný generátor. Ten sa skladá z posuvného registra a akumulátora. Dizajnovo sa jedná o Grain-128a [2], ktorý bol rozšírený o podporu AEAD a umožnenie väčších autentifikátorov. Chcel by som podotknúť, že pracujeme nad GF(2), tak operáciou + myslíme operáciu XOR, ktorú ale značíme ako +.

2.1 Stavebné bloky a funkcie

Prvý blok, generátor predvýstupu, najskôr vygeneruje prúd pseudonáhodných bitov $y_{512+t}, 0 \leq t \leq L$, kde L je dĺžka požadovaného hesla šifry. Tieto bity sa následne použijú pri šifrovaní a pre autentifikačnú značku. Znázornenie blokov môžeme vidieť na obrázku 2.1. Obidva registre sú 128



Obr. 2.1 Stavebné bloky v Grain-128AEADv2 [1]

bitové. LFSR budeme označovať ako $S_t = [s_0^t, s_1^t, \ldots, s_{127}^t]$ a NFSR ako $B_t = [b_0^t, b_1^t, \ldots, b_{127}^t]$. Vnútorný stav generátora predvýstupu je reprezentovaný pravé týmito dvoma registrami. Spolu tvoria vnútorný stav o veľkosti 256 bitov.

Zavedieme si primitívny spätnoväzbový polynóm 1.6 nad GF(2), ktorý budeme označovať ako *f(x)*. Je to charakteristický polynóm, ktorý definujeme nasledovne

$$f(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128}.$$

Platí, že ak charakteristický polynóm $f(x)$ registru R je primitívny, tak každý z jeho $2^n - 1$ nenulových inicializačných vnútorných stavov produkuje výstupnú postupnosť maximálnej možnej periódy $2^n - 1$, kde n je dĺžka registru. To je hlavný dôvod, prečo má byť polynóm primitívny.

Prislúchajúca funkcia pre aktualizáciu LFSR je definovaná ako

$$\begin{aligned} s_{127}^{t+1} &= s_0^t + s_7^t + s_{38}^t + s_{70}^t + s_{81}^t + s_{96}^t \\ &= \mathcal{L}(S_t). \end{aligned}$$

Pre NFSR je nelineárny spätnoväzbový polynóm *g(x)* nad GF(2) definovaný nasledovne

$$\begin{aligned} g(x) = 1 &+ x^{32} + x^{37} + x^{72} + x^{102} + x^{128} + x^{44}x^{60} + x^{61}x^{125} \\ &+ x^{63}x^{67} + x^{69}x^{101} + x^{80}x^{88} + x^{110}x^{111} + x^{115}x^{117} \\ &+ x^{46}x^{50}x^{58} + x^{103}x^{104}x^{106} + x^{33}x^{35}x^{36}x^{40} \end{aligned}$$

a funkcia pre aktualizáciu ako

$$\begin{aligned} b_{127}^{t+1} &= s_0^t + b_0^t + b_{26}^t + b_{56}^t + b_{91}^t + b_{96}^t + b_3^t b_{67}^t + b_{11}^t b_{13}^t \\ &+ b_{17}^t b_{18}^t + b_{27}^t b_{59}^t + b_{40}^t b_{48}^t + b_{61}^t b_{65}^t + b_{68}^t b_{84}^t \\ &+ b_{22}^t b_{24}^t b_{25}^t + b_{70}^t b_{78}^t b_{82}^t + b_{88}^t b_{92}^t b_{93}^t b_{95}^t \\ &= s_0^t + \mathcal{F}(B_t). \end{aligned}$$

Bit s_{127}^{t+1} a b_{127}^{t+1} budeme ďalej nazývať ako aktualizačný bit pre LFSR a NFSR.

Vstupom boolovskej funkcie $h(x)$ je deväť stavových premenných. Dva bity sú prevzaté z NFSR a sedem z LFSR. Daná funkcia je definovaná ako

$$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8.$$

Zodpovedajúcim premenným $x_0, \ldots, x_8$ prislúchajú stavové premenné $b_{12}^t, s_8^t, s_{13}^t, s_{20}^t, b_{95}^t, s_{42}^t, s_{60}^t, s_{79}^t$ a s_{94}^t.

Výstupom celého prvého hlavného bloku, generátora predvýstupu, je funkcia predvýstupu, ktorá je daná ako

$$y_t = h(x) + s_{93}^t + \sum_{j \in \mathcal{A}} b_j^t,$$

kde $\mathcal{A} = \{2, 15, 36, 45, 64, 73, 89\}$.

Posledný blok, autentifikačný generátor, pozostáva už zo spomínaného posuvného registra, ktorý obsahuje posledných 64 bitov z predvýstupu a akumulátora. Obidva sú o veľkosti 64 bitov. Posuvný register uchováva najnovších 64 nepárnych bitov z predvýstupu. Obsah akumulátora v inštancii i budeme označovať ako $A_i = [a_0^i, a_1^i, \ldots, a_{63}^i]$ a shift registru ako $R_i = [r_0^i, r_1^i, \ldots, r_{63}^i]$.

2.2 Inicializácia kľúča a nonces

Pred generovaním keystreamu a autentifikácie potrebujeme mať vnútorný stav generátora predvýstupu. V tejto kapitole sa pozrieme na to, ako inicializovať vnútorný stav generátora predvýstupu a autentifikačného generátora. Bity kľúča budeme značiť k_i, kde $0 \leq i \leq 127$ a bity nonce ako IV_i,

pre $0 \leq i \leq 95$. Potom inicializácia prebieha načítaním 128 bitov NFSR z bitov kľúča $b_i^0 = k_i$, pre $0 \leq i \leq 127$ a prvých 96 prvkov LFSR pomocou bitov z nonce, $s_i^0 = IV_i$, kde $0 \leq i \leq 95$. Posledných 32 bitov LFSR sa zaplní jednotkami $s_i^0 = 1$, kde $96 \leq i \leq 126$ a nulou na konci $s_{127}^0 = 0$. Nasledovne sa v šifre 320-krát XORuje vnútorný stav funkcie predvýstupu so vstupom do LFSR a NFSR, t.j.,

$$s_{127}^{t+1} = \mathcal{L}(S_t) + y_t, \quad 0 \leq t \leq 319$$
$$b_{127}^{t+1} = s_0^t + \mathcal{F}(B_t) + y_t, \quad 0 \leq t \leq 319.$$

Potom sa znovu použije kľúč a ten sa XORuje 64-krát so vstupom LFSR a NFSR, t.j.,

$$s_{127}^{t+1} = \mathcal{L}(S_t) + y_t + k_{t-256}, \quad 320 \leq t \leq 383$$
$$b_{127}^{t+1} = s_0^t + \mathcal{F}(B_t) + y_t + k_{t-320}, \quad 320 \leq t \leq 383.$$

Teraz si ukážeme, ako sa inicializuje autentifikačný generátor. Keystream predvýstupu použijeme na načítanie registra a akumulátora nasledovne

$$a_j^0 = y_{384+j}, \quad 0 \leq j \leq 63$$
$$r_j^0 = y_{448+j}, \quad 0 \leq j \leq 63.$$

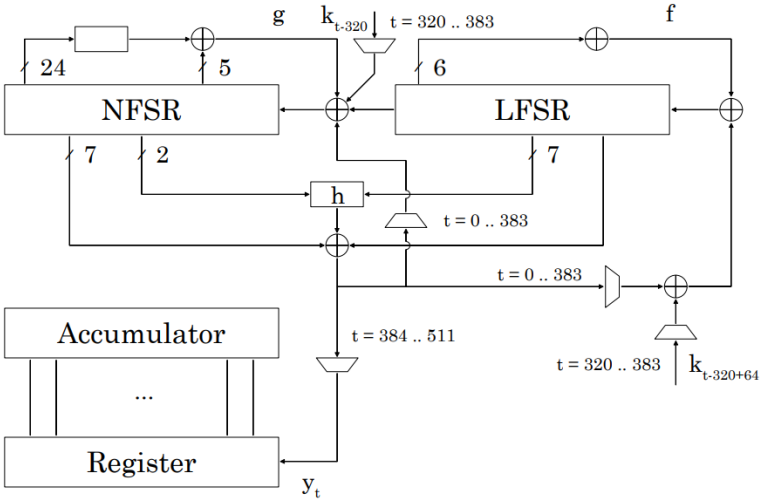
Pri inicializácii registra a akumulátora sa súčasne aktualizuje LFSR ako

$$s_{127}^{t+1} = \mathcal{L}(S_t), \quad 384 \leq t \leq 511$$

a taktiež aj NFSR

$$b_{127}^{t+1} = s_0^t + \mathcal{F}(B_t), \quad 384 \leq t \leq 511$$

Postup inicializácie môžeme vidieť na obrázku 2.2 a pre lepšie znázornenie aj v algoritme 4. Po inicializácii šifry sú vnútorné stavy LFSR a NFSR označené ako S_{512} a B_{512} a register s akumulátorom ako R_0 a A_0.



Obr. 2.2 Prehľad inicializácie v Grain-128AEADv2 [1]

Algoritmus 4 Inicializácia kľúča a nonces

Vstup: $k - kľúč, IV - nonce$
Výstup: $b - NFSR, s - LFSR$

//Naplnenie NFSR a LFSR prostredníctvom k a nonces
for $i \leftarrow 0$ to 127 **do**
    $b_i^0 = k_i$
    if $0 \leq i \leq 95$ **then**
        $s_i^0 = IV_i$
    end if
    if $96 \leq i \leq 126$ **then**
        $s_i^0 = 1$
    end if
end for
$s_{127}^0 = 0$
//XOR 320-krát vnútorného stavu funkcie predvýstupu so vstupom do NFSR a LFSR
for $t \leftarrow 0$ to 319 **do**
    $s_{127}^{t+1} = \mathcal{L}(S_t) + y_t$
    $b_{127}^{t+1} = s_0^t + \mathcal{F}(B_t) + y_t$
end for
//XOR 64-krát kľúča k so vstupom do NFSR a LFSR
for $t \leftarrow 320$ to 383 **do**
    $s_{127}^{t+1} = \mathcal{L}(S_t) + y_t + k_{t-256}$
    $b_{127}^{t+1} = s_0^t + \mathcal{F}(B_t) + y_t + k_{t-320}$
end for
//Inicializácia autentifikačného generátora
for $j \leftarrow 0$ to 63 **do**
    $a_j^0 = y_{384+j}$
    $r_j^0 = y_{448+j}$
end for
for $t \leftarrow 384$ to 511 **do**
    $s_{127}^{t+1} = \mathcal{L}(S_t)$
    $b_{127}^{t+1} = s_0^t + \mathcal{F}(B_t)$
end for

2.3 Prevádzkový režim

Majme správu m o dĺžke L, ktorá má prvky $m_0, m_1, \ldots, m_{L-1}$, a nastavme $m_L = 1$ ako padding, aby sa zabezpečilo, že m a $m\|0$ majú rôzne značky.

Po inicializácii generátora predvýstupu, ktorú sme si ukázali v minulej podkapitole 2.2, sa jeho výstup použije na generovanie bitov keystreamu z_i a autentifikačných bitov z_i'. Keystream sa používa pri šifrovaní a autentifikačné bity na aktualizáciu registra v akumulačnom generátore. Keystream sa generuje ako

$$z_i = y_{512+2i}.$$

Každý párny bit z generátora predvýstupu sa použije pre bity keystreamu pričom sa prvých 512 bitov neuvažuje. Naopak pre autentifikáciu sa použije každý nepárny bit nasledovne

$$z_i' = y_{512+2i+1}.$$

Správa sa šifruje ako

$$c_i = m_i + z_i, \quad 0 \leq i < L.$$

Akumulátor sa aktualizuje pomocou nasledujúceho vzorca

$$a_j^{i+1} = a_j^i + m_i r_j^i, \quad 0 \leq j \leq 63, \quad 0 \leq i \leq L$$

a posuvný register ako

$$r_{63}^{i+1} = z_i'$$
$$r_j^{i+1} = r_{j+1}^i, \quad 0 \leq j \leq 62$$

2.4 Obmedzenie kľúča

Vďaka tomu ako boli navrhnuté šifry z rodiny Grain, je možné šifrovať veľké množstvo údajov pomocou rovnakého páru kľúč/nonce. Pri verzii 2, Grain-128AEADv2, autori obmedzili počet bitov toku kľúčov pre každý kľúč/nonce na 2^{80}. To znamená, že môžeme vygenerovať maximálne 2^{81} bitov predvýstupu pre jeden pár kľuč/nonce.

2.5 Autentizované šifrovanie s asociovanými dátami

Grain-128AEADv2 umožňuje autentizované šifrovanie s asociovanými dátami (anglicky Authetication Encryption with Asociated Data, skrátene AEAD) pomocou explicitného vynútenia y_{512+2i} na nulu pre bity, ktoré by nemali byť šifrované ale mali by sa dať overiť. Zadefinujeme si nasledovnú AEAD masku

$$d = d_0, d_1, \ldots, d_{L-1},$$

ktorá nám udáva aké bity sa majú šifrovať. Ak maska obsahuje samé jednotky, t.j. nemáme žiadne asociované dáta, šifrovanie prebieha ako je uvedené vyššie. Ak by sme nemali samé jednotky v maske, tak sa šifruje nasledovne

$$c_i = m_i + z_i \cdot d_i, \quad 0 \leq i < L,$$

Maska AEAD by mala byť vopred určená a nemenná pre protokol používajúci Grain-128AEADv2. Táto maska zahŕňa skvelú flexibilitu, keďže nešifrované dáta môžu byť na ľubovolných pozíciách paketu. Nevýhodou je, že po zašifrovaní sú niektoré bity predvýstupu nevyužité, t.j. nepoužijú sa pri šifrovaní. Autori uvádzajú, že aj tak ide o efektívne riešenie, keďže v podstate nie sú žiadne dodatočné náklady na uvedenie nezašifrovaných dát v konštrukcii.

Ak by sme mali asociované dáta s premenlivou dĺžkou, ako v špecifikovanom aplikačnom programovacom rozhraní Národného inštitútu pre štandardy a technológie (anglicky National Institute of Standards and Technology Application Programming Interface, skrátene NIST API), tak by sa prvých X bitov dát nemalo zašifrovať a zvyšných $L - X$ by sa malo zašifrovať. Potom $d_i = 0$, ak $i < X$ a inak $d_i = 1$ a následne sa vygeneruje zašifrovaný text.

2.6 Grain-128AEADv2 s NIST API

Teraz sa pozrieme ako použiť algoritmus Grain-128AEADv2 v NISTs API. Premennou m budeme rozumieť správu odosielanú do API, ktorá sa bude šifrovať a premennou m' úplný reťazec, ktorý je použitý v algoritme Grain-128AEADv2. Podobne definujeme šifrovaný text c, ktorý obsahuje zašifrovaný reťazec a informáciu o autentifikácii a taktiež c' bitový reťazec dešifrovaný algoritmom Grain-128AEADv2. Keďže NIST API je bajtovo orientované, takže padding sa označuje ako $0x80$ namiesto "1". Tieto dva paddingy sú navzájom ekvivalentné.

2.6.1 Šifrovanie a generovanie MAC s NIST API

NIST API má špecifický prístup, preto budeme mapovať vstup po bajtoch (asociované dáta, dĺžka asociovaných dát, správa, dĺžka správy) do bitového reťazca m' nasledovne

$$m' = Encode(dĺžka\ asociovaných\ dát)\|asociované\ dáta\|správa\|0x80,$$

kde Encode() = y označujeme kódovanie dĺžky podobné definitnej forme použitej v kódovaní DER. Zoberieme si prvý bajt y, ak tento bajt začína nulou tak zvyšných 7 bitov je kódovanie počtu bajtov v asociovaných dátach. Ak naopak bude prvý bajt začínať jednotkou, zvyšných 7 bitov je zakódovanie počtu nadchádzajúcich bajtov, ktoré popisujú dĺžku asociovaných dát. Po prvom bajte y nasledujú bajty opisujúce dĺžku.

Po správnom mapovaní vstupu zašifrujeme a autentifikujme m' v AEAD móde nastavením masky AEAD tak, že $d_i = 0$ pre všetky bity $i < M$, kde M označuje bitovú dĺžku reťazca $Encode(dĺžka\ asociovaných\ dát)\|(asociované\ dáta)$. Naopak pre $i \geq M$ nastavíme $d_i = 1$. Spôsob tohto šifrovania môžeme vidieť aj v algoritme 5.

Algoritmus 5 Šifrovanie s NIST API

Vstup: $ad-asociované\ dáta, adlen-dĺžka\ asociovaných\ dát, m-správa, mlen-dĺžka\ správy, k-kľúč, nonce$
Výstup: $c-šifrovaná\ správa$

1: Inicializuj generátor s k a $nonce$
2: Zostroj $m' = (Encode(adlen)\|ad\|m\|0x80)$
3: $M =$ bitová dĺžka $Encode(adlen)\|ad$
4: $d_i = 0, \quad (0 \leq i \leq M-1)$
5: $d_i = 1, \quad (M \leq i \leq M + mlen - 1)$
6: **Šifruj** pomocou $c'_i = m'_i + z_i d_i, \quad (0 \leq i \leq M + mlen - 1)$
7: **Over** pomocou z'_i a generuj $A_{M+mlen+1}$
8: $c = (c'_M, c'_{M+1}, \ldots, c'_{M+mlen-1})\|A_{M+mlen+1}$
9: return 0

2.6.2 Dešifrovanie a overenie MAC s NIST API

Prijímateľ musí overiť overovací kód správy (anglicky Message Authentication Code, skrátene MAC) a vykonať dešifrovanie. Vstupom API sú asociované dáta a šifrovaný text c. Šifrovaný text c tvorí zašifrovaná správa spojená s MAC. Pomocou kódovania ad length a paddingu vytvoríme bitový string c'. Bitový reťazec m' vypočítame ako $m'_i = c'_i + z_i \cdot d_i$, pre $\quad 0 \leq i < L$. Tento krok taktiež zahŕňa prepočítanie MAC z m' podobne ako v šifrovaní. Následne porovnáme dve 64 bitové hodnoty MAC, výsledkom čoho je flag hodnota, ktorá nadobúda hodnotu 0 ak sú obidve MAC rovnaké, inak obsahuje hodnotu -1. Toto porovnanie je súčasťou základu implementácie. Môžeme vykonávať aj ďalšie možné kontroly, ktoré nie sú závislé od hlavnej implementácie. Napr. kontrola opätovného použitia nonce. Tá závisí od aplikácie a preto sa uskutočňuje mimo hlavnej implementácie. Dešifrovanie a overenie MAC pomocou AEAD je vidno v algoritme 6.

Algoritmus 6 Dešifrovanie s NIST API

Vstup: *ad − asociované dáta, adlen − dĺžka asociovaných dát, c − šifrovaná správa, clen − dĺžka šifrovanej správy, k − kľúč, nonce*
Výstup: *m − správa*

Inicializuj generátor s k a *nonce*
Zostroj $c' = (Encode(adlen) \| ad \| c_0, \ldots, c_{clen-65} \| 0x80)$
Nech
  $M =$ bitová dĺžka $Encode(adlen) \| ad$
  $mlen = clen - 64$
  $d_i = 0, \quad (0 \leq i \leq M - 1)$
  $d_i = 1, \quad (M \leq i \leq M + mlen - 1)$
Dešifruj pomocou $m'_i = c'_i + z_i d_i, \quad (0 \leq i \leq M + mlen - 1)$
Over pomocou z'_i a generuj $A_{M+mlen+1}$
Nastav $m = m'_M, \ldots, m'_{M+mlen-1}$
if $(c_{clen-64}, \ldots, c_{clen-1}) == A_{M+mlen+1}$ **then**
  return 0
else
  return -1

Kapitola 3

Súvisiace práce

Táto kapitola sa venuje popisu článkov súvisiacich s danou témou diplomovej práce. Dozviete sa napríklad informácie o už vykonaných útokoch na šifry z rodiny Grain.

3.1 Algebraické útoky na generátory hesla šifier Grain

Tento článok [11] sa zaoberá algebraickými útokmi na niektoré generátory kľúčových prúdov, ktoré pozostávajú z NFSR, LFSR a filtrovacej funkcie. Autori vysvetľujú, že ak sa nesprávne zvolí filtrovacia funkcia, dajú sa na daný generátor použiť nové algebraické útoky využívajúce prístup rozdeľuj a panuj. Týmto prístupom sa dá obnoviť počiatočný vnútorný stav LFSR a následne aj NFSR.

Analýza algebraických útokov na modifikované šifry Grain ukázala, že napriek vysoko nelineárnym filtračným funkciám sa vnútorný stav LFSR dá obnoviť výrazne rýchlejšie ako pri vyskúšaní všetkých možností kľúčov. Následne po obnovení vnútorného stavu LFSR je možné obnoviť vnútorný stav NFSR ale len čiastočne. Zvyšok vnútorného stavu sa doplní skúšaním všetkých možností. Týmto útokom autori poukazujú na zraniteľnosť týchto generátorov hesiel so štruktúrou podobnou ako Grain a zdôrazňujú dôležitosť správneho zvolenia filtrovacej funkcie.

3.2 Diferenciálny chybový útok na prúdové šifry z rodiny Grain

V tomto článku [12] autori študujú diferenciálny chybový útok založený na určitých vlastnostiach boolovských funkcií a zodpovedajúcom výbere bitov vo vnútornom stave LFSR, ktoré ovplyvňujú vstup. Tento útok autori použili na prúdové šifry z rodiny Grain. Ukazujú, že diferenciálny chybový útok možno efektívne použiť na boolovskú funkciu použitú v Grain v1. Ich myšlienkou je nájsť vhodné α, že $h(x) + h(x + \alpha)$ je lineárne. Následne použijú metódy na identifikáciu chybových miest a zostavenie lineárnych rovníc pre získanie obsahu LFSR a NFSR. Autori navrhli presné kritéria pre návrh danej boolovskej funkcie. Šifry s podobnou štruktúrou ako pri šifrách z rodiny Grain, môžu použiť tento návrh ako protiopatrenie proti danému typu útoku.

3.3 Rýchle korelačné útoky na prúdové šifry podobné Grain šifrám s malým vnútorným stavom

Jednou z klasických kryptoanalytických technik proti prúdovým šifrám založeným na LFSR je útok typu rýchlej korelácie. V danom článku [13] autori študujú bezpečnosť prúdových šifier s malým vnútorným stavom [14] (anglicky Small-state stream ciphers, skrátene SSC), so štruktúrou podobnou šifrám z rodiny Grain, práve pomocou daného typu útoku. Pre úschovu bitov kľúča v kryptosystéme po inicializácii boli navrhnuté SSC. Tieto bity kľúča sa uschovávajú aby ich bolo možné opätovne použiť pre rôzne počiatočné hodnoty. Preto SSC navrhnuté tak, aby mali výrazne menšiu veľkosť a nízku spotrebu energie. Pri podmienke, že nelineárna kombinovaná funkcia spĺňa nejakú charakteristiku, dokážu autori rozšíriť kaskádovú štruktúru a obnoviť úplný vnútorný stav po častiach. Pre príklad používajú daný typ útoku proti vylepšenej verzii Sprout-u nazvanou Fruit [15]. Fruit využíva aktualizáciu vnútorného stavu závislú od kľúča vo fáze generovania kľúčového prúdu.

3.4 Algebraické útoky a rozklad booleovských funkcií

Algebraické útoky na prúdové šifry založené na LFSR využívajú mnohorozmerné vzťahy zahŕňajúce bity kľúča a výstupné bity. Pri nájdení takýchto vzťahov nízkych stupňov sa útok stáva veľmi efektívnym. Pri takomto type algebraickom útoku sa využívajú nízke násobky booleovských funkcií vytvorené pri návrhu prúdových šifier. V článku [16] sa autori zaoberajú práve problémom násobkov booleovských funkcií. Autori dokázali odhadnúť pravdepodobnosť, že booleovská funkcia má násobok nízkeho stupňa. Následne zostrojili nový algoritmus, ktorý umožňuje rozhodnúť, či má booleovská funkcia nízke násobky stupňa. Týmto umožnili vytvoriť kritérium, ktoré dokáže preukázať bezpečnosť voči algebraickým útokom. Týmto algoritmom ukázali, že nízke násobky stupňa má aj trieda stupňov optimalizovaných funkcií Maiorana-McFarland [17].

3.5 Podmienené diferenciálne útoky na prúdovú šifru Grain-128a

Autori pre porovnanie použili podmienený diferenciálny útok navrhnutý Michaelom Lehmannom. Ten v analýze navrhol rozlišovací útok na 189 kôl šifry Grain-128a so slabým kľúčovým nastavením. V článku [18] sú opísané dva podmienené diferenciálne útoky na šifru Grain-128a. V prvom z nich dokázali autori získať pre 169-kôl Grain-128a 18 tajných kľúčových výrazov. Kľúčový výraz je hodnota alebo pole hodnôt, ktoré sa rovnajú príslušným bitom kľúča.Pre redukovaný Grain-128a dokázal tento útok ako prvý získať tajné výrazy kľúča. Následne v druhom útoku dokázali rozšíriť už spomínaný rozlišovací útok na Grain-128a so slabým kľúčom až na 195-kôl. Týmto autori navrhli najznámejší útok pre redukovaný Gran-128a, čo sa týka počtu napadnutých kôl.

3.6 Efektívne algoritmy na riešenie predefinovaných systémov viacrozmerných polynomických rovníc

Mnohé kryptosystémy sú založené na probléme riešenia veľkých systémov viacrozmerných kvadratických polynomických rovníc nad konečným poľom. Riešenie tohto problému je NP-úplný problém a obecne má dvojnásobne exponenciálnu zložitosť nad akýmkoľvek poľom. Pre rovnaký počet rovníc m a neznámych n sa používajú Grobnerove bázy. V tomto článku [19] autori

analyzovali algoritmus relinearizácie a následne vytvorili nový algoritmus, ktorý nazvali XL algoritmus. Tento algoritmus je jednoduchší a výkonnejší ako relinearizácia. Majme ϵ, pre ktoré platí $0 < \epsilon \leq 1/2$ a majme $m \geq \epsilon n^2$. Autori ukázali, že pri týchto podmienkach XL a relinearizácia prebehnú v polynomiálnom čase približne $n^{O(1/\sqrt{\epsilon})}$.

3.7 Vylepšené prehľadávanie všetkých možností kľúčov pri útoku na prúdové šifry

Autori sa v danom článku [20] venujú prehľadávaniu všetkých možných stavov generátora hesiel, aby dokázali určiť vnútorný stav tohto generátoru. Pri prehľadávaní kontrolujú výsledné a pozorované heslo. Zložitosť dvoch útokov, ktoré autori ukazujú, rastie exponenciálne s veľkosťou stavu generátora. Napriek tomu sú efektívnejšie ako jednoduché prehľadávanie všetkých stavov generátora. Ak máme dostatočné úložisko a dostatočne známe heslo, potom podľa autorov je možné pri daných útokoch znížiť efektívnu entropiu prehľadávaného stavu na polovicu.

Kapitola 4

Implementácia

V tejto kapitole sa budeme venovať jedným z hlavných krokov a to popisu implementácie prevodu šifry Grain128AEADv2, ďalej budeme pre jednoduchosť nazývať ako Grain, na sústavu polynomických rovníc, jej následnému riešeniu a postupu zmenšovania šifry. Celý postup je rozdelený do štyroch skriptov:

- V **Grain.py** sa nachádza implementácia šifry Grain, ktorou následne vygenerujeme potrebný numerický keystream.
- Prostredníctvom skriptu **GrainPolynomial.sage** sa šifra prevedie na sústavu polynomických rovníc.
- Skript **MagmaSolver.py** slúži na vytvorenie magma skriptu, ktorý nájde Groebnerovu bázu 1.27 a riešenie výslednej sústavy.
- K vykonaniu testov na danej implementácii slúži skript **Tests.py**.

4.1 Potrebný software

Využitý software použitý pre implementáciu šifry bol nasledujúci:

- Python 2.7.18,
- SageMath 9.0,
- Magma V2.25-5.

4.1.1 Magma

Magma je softvérový balík, ktorý poskytuje prostredie pre výpočty v rôznych odvetviach matematiky ako je napríklad algebra, teória čísel, algebraická kombinatorika a tak ďalej. V našej práci hlavne využijeme implementáciu algoritmu pre riešenie sústavy polynomiálnych rovníc a možnosť vypočítať Groebnerovu bázu polynomiálneho ideálu. Vychádzali sme z dokumentácie, ktorá je k dispozícii z [21].

4.2 Zjednodušené varianty šifry Grain

Algebraickú kryptoanalýzu budeme aplikovať na zmenšené varianty šifry Grain. V tejto podkapitole si povieme ako sme zmenšovali danú šifru. K zmenšeniu sme použili nasledujúce parametre:

- n - bitová verzia šifry,
- r - celkový počet kôl v danej šifre,

Bitová verzia šifry je definovaná parametrom n, ktorý udáva veľkosť NFSR. Keďže NFSR je inicializované celkovým obsahom kľúča, tak veľkosť kľúča je taktiež definovaná parametrom n. Jednotlivé polynómy z výslednej sústavy budú obsahovať premenné $b_0, b_1, \ldots, b_{n-1}$, ktorých hodnoty sa budeme snažiť nájsť. Tieto premenné sú pri inicializácii vložené do indexov NFSR. Pri jednotlivých kolách sa premenné sčítajú alebo prenásobia a tým vzniknú nové polynómy uložené v NFSR.

Parameter n bude taktiež ovplyvňovať veľkosti vygenerovaných indexov zo spätnoväzbových polynómov pri jednotlivých metódach triedy GrainPolynomial 4.3.2.

▶ **Príklad 4.1.** Majme napríklad indexy [96,81,70,38,7,0] vygenerované z primitívneho spätnoväzbového polynómu určené na výpočet aktualizačného bitu, respektíve polynómu pre LFSR:

$$f(x) = 1 + x^{32} + x^{47} + x^{58} + x^{90} + x^{121} + x^{128}.$$

Z daných indexov ponecháme len tie, ktoré sú menšie ako je parameter n. Dôvodom je zmenšenie veľkosti LFSR v závislosti od parametru n. Tým pádom pre napríklad 16 bitovú verziu dostaneme indexy [7,0].

Podobne to bude platiť aj pre indexy NFSR alebo pri funkcii predvýstupu y. Zmenšenie booleovskej funkcie h:

$$h(x) = x_0x_1 + x_2x_3 + x_4x_5 + x_6x_7 + x_0x_4x_8,$$

je implementované tak, že sa ponechajú len tie súčiny, ktorých jednotlivé indexy prvkov sú menšie ako je parameter n. Takže pre premenné

$$(x_0, \ldots, x_8) = (b^t_{12}, s^t_8, s^t_{13}, s^t_{20}, b^t_{95}, s^t_{42}, s^t_{60}, s^t_{79}, s^t_{94}),$$

pri použití parametru $n = 16$ budeme v booleovskej funkcii uvažovať len súčin $b^t_{12} \cdot s^t_8$.

Celkový počet kôl je zadaný parametrom r. Z parametru r sa vypočítajú počty kôl pre jednotlivé fázy inicializácie 4.3.2.2:

- počet kôl pre metódu `round_init` je $3n - n/2$.
- počet kôl metódy `round_init_with_key` je $n/2$.
- počet kôl pri inicializácii registra R a následne aj akumulátora A pomocou metódy `round_` je $n/2$.

▶ **Príklad 4.2.** Majme plnú 128 bitovú verziu šifry. Pre ňu platí, že počet kôl pre jednotlivé metódy vyplývajúcich z kapitoly teoretického základu 2 je nasledovný:

- Pre `round_init` je počet kôl rovný 320.
- Pre `round_init_with_key` zase 64 kôl (polovica veľkosti kľúča).
- Pre `round_` je počet kôl rovný $2 \cdot 64$ (jeden krát pre R a druhý krát pre A).

Celkový počet kôl je teda $r = 512$.

Nech $n = r/4 = 512/4 = 128$, potom pomocou vyššie definovaných rovníc v príklade 4.1 dostaneme $3n - n/2 = 3 \cdot 128 - 128/2 = 320$, $n/2 = 128/2 = 64$ a $n/2 = 128/2 = 64$, čo sú presne počty kôl pri plnej 128 bitovej verzii.

▶ **Príklad 4.3.** Z predchádzajúceho príkladu pre $n = r/4 = 512/4 = 128$ je vidno, že daným výpočtom dostaneme priamo bitovú verziu. Takže ak uvažujeme 16 bitovú verziu šifry Grain, tak potom bude parameter n = 16 a jednotlivé kolá sú $3n - n/2 = 3 \cdot 16 - 16/2 = 40$, $n/2 = 16/2 = 8$ a $n/2 = 16/2 = 8$. Takže celkový počet kôl pre 16 bitovú verziu je 40 + 8 + 8 (akumulátor A) + 8 (register R) = 64

▶ **Príklad 4.4.** Ak by sme chceli vytvoriť zmenšenú variantu *Grain*(n = 16,r = 8), kde $n = r/4 = 8/4 = 2$, potom jednotlivý počet kôl je $3n - n/2 = 3 \cdot 2 - 2/2 = 5$, $n/2 = 2/2 = 1$ a $n/2 = 2/2 = 1$ a celkový počet kôl je teda 5 + 1+ 1 (akumulátor A) + 1 (register R) = 8

4.3 Skript GrainPolynomial.sage

Ako sme už spomínali, tento skript je podobný skriptu Grain.py, v ktorom ide o implementáciu teoretického popisu šifry 2. Rozdiel je v tom, že výsledný keystream nebude numerický ale v podobe sústavy rovníc. Preto sa nebudeme ďalej venovať implementácii šifry v skripte Grain.py a rovno sa pozrieme na princíp prevodu šifry na sústavu polynomiálnych rovníc.

4.3.1 Main

Skript sa dá spustiť príkazom:

```
sage GrainPolynomial.sage bit_version number_of_rounds.
```

Prostredníctvom parametrov zadáme, o akú bitovú verziu sa jedná a aký má byť celkový počet kôl pre danú šifru. Po načítaní parametrov sa nasledovne vytvorí booleovský polynomický okruh 1.12, ktorý definuje rovnice nad poľom *GF(2)*. Tento okruh je definovaný s premennými $b_0, b_1, \ldots, b_{n-1}$, kde n je bitová verzia šifry. Potom sa vytvorí inštancia triedy *GrainPolynomial* 4.3.2 a pomocou nej sa vygeneruje keystream. Pre potreby testovania sa zapíše čas a využitá pamäť počas generovania rovníc do prislúchajúceho súboru.

4.3.2 Trieda GrainPolynomial

Trieda slúži k prevodu danej šifry na sústavu polynomických rovníc. Hodnoty vnútorných stavov NFSR tvoria polynómy nad poľom GF(2). Počiatočné hodnoty LFSR sú inicializované hodnotami z nonce kvôli zvyšujúcej sa zložitosti generovania rovníc. V tejto kapitole boli opísané len vybrané metódy.

4.3.2.1 Konštruktor

Argumenty:

- `bit_version` - bitová verzia šifry
- `number_of_rounds` - počet kôl šifry

V inicializačnej fázi konštrutkoru sa NFSR naplní premennými $b_0, b_1, \ldots, b_n$, kde n je bitová verzia šifry. Následne sa vypočíta počet jednotlivých kôl pre rôzne typy inicializačných fáz, ktoré sú spomínané v podkapitole o zmenšených variantách šifry 4.2. Po výpočte kôl sa spustia metódy, ktoré sú implementované podľa troch fáz inicializácie. Sú to metódy `round_init`, `round_init_with_key`, ktoré sú uvedené v 4.3.2.2. Posledná fáza je inicializácia posuvného registra R a akumulátora A pomocou metódy `round_`.

4.3.2.2 Metódy inicializácie

Najprv sa v jednotlivých metódach vypočíta aktualizačný polynóm pre LFSR a NFSR. Aktualizačný polynóm je definovaný rovnako ako aktualizačný bit, až na to, že jeho hodnota je polynóm. Výpočet tohto polynómu je uvedený v 4.3.2.3. Potom sa vypočíta výstup z funkcie y uvedenej v 4.3.2.4. Po posunutí registrov sa na ich jednotlivé posledné bity vloží výsledný polynóm podľa použitých metód inicializácie nasledovne:

- `round_init` - inicializačné kolá, v ktorých výsledný polynóm je XOR aktualizačného bitu s funkciou y.
  - LFSR - `next_bit_lfsr + y`
  - NFSR - `next_bit_nfsr + y`
- `round_init_with_key` - inicializačné kolá so XORom funkcie y a bitom kľúča. Premennou i označujeme aktuálne kolo. Pri tejto metóde sa vždy pripočíta k LFSR prvá polovica kľúča a k NFSR druhá.
  - LFSR - `next_bit_lfsr + y + key[i + bit_version/2]`
  - NFSR - `next_bit_nfsr + y + key[i]` kde i je aktuálne kolo.
- `round_` - počiatočná metóda pre aktualizáciu posuvných registrov. Táto metóda sa používa pri naplnení akumulátora, posuvného registra a taktiež pri generovaní keystreamu.
  - LFSR - `next_bit_lfsr`
  - NFSR - `next_bit_nfsr`

4.3.2.3 Metódy update_bit_lfsr a update_bit_nfsr

Aktualizačný polynóm pre LFSR sa vypočíta pomocou indexov vytvorených z primitívneho spätnoväzbového polynómu a taktiež aj aktualizačný polynóm pre NFSR pomocou indexov z jeho prislúchajúceho spätnoväzbového polynómu z kapitoly 2. Jednotlivé indexy sú ešte pred výpočtom upravené podľa toho o akú zmenšenú variantu šifry sa jedná. Potom sa prechádza cez NFSR, respektíve LFSR a do výsledného aktualizačného polynómu sa ukladajú jednotlivé sčítania polynómov, respektíve násobky polynómov pri NFSR, uložených na daných indexoch.

▶ **Príklad 4.5.** Majme následujúce indexy pre LFSR [96,81,70,38,7,0]. Pre 16 bitovú verziu budeme teda uvažovať len index 7 a 0. Na týchto indexoch sú uložené polynómy

$$b_0 + b_4,$$

$$b_0 + b_1 + b_3 + b_4.$$

Výsledný aktualizačný polynóm po redukovaní modulom 2 je

$$\begin{aligned} next\_bit\_lfsr &= LFSR[0] + LFSR[7] \\ &= b_1 + b_3 \end{aligned}$$

▶ **Príklad 4.6.** Majme následujúce indexy [[0],[3],[11,13]] pre NFSR ak počítame so 16 bitovou verziou šifry. Vnorené indexy reprezentujú súčin medzi hodnotami na daných indexoch. Takéto súčiny následne sčítame. Na indexoch sú nasledujúce polynómy:

$$b_4$$

$$b_4 + b_5$$

$$b_{10} + b_{11}$$

$$b_4 + b_{10}$$

Výsledný aktualizačný polynóm po redukovaní modulom 2 je

$$\begin{aligned} next\_bit\_nfsr &= NFSR[0] + NFSR[3] + NFSR[11] \cdot NFSR[13] \\ &= b_5 + b_{10} + b_4 b_{10} + b_{10} b_{11} + b_4 b_{11} \end{aligned}$$

4.3.2.4 Metódy function_h a function_y

Funkciu h tvorí súčet súčinov jednotlivých polynómov na definovaných indexoch z LFSR a NFSR. Indexy sa redukujú podľa aktuálnej bitovej verzie šifry. Funkciu y zase tvorí súčet výsledného polynómu z funkcie h, polynóm z LFSR na indexe 93 a súčet polynómov z NFSR podľa definovanej množiny $\mathcal{A}$ 2. Použijú sa iba indexy menšie než je veľkosť NFSR.

▶ **Príklad 4.7.** Nech [2,15] sú indexy z redukovanej množiny $\mathcal{A}$. Funkcia h je pri 16 bitovej verzii šifry definovaná nasledovne

$$h(x) = NFSR[8] \cdot LFSR[12].$$

Majme zoradené polynómy podľa indexov od najmenšie po najväčší nasledovne:

$$b_3$$

$$b_8$$

$$b_8 + b_{10}$$

$$b_2 + b_3$$

Výsledný výstupný polynóm z funkcie y po redukovaní modulom 2 je

$$\begin{aligned} y &= NFSR[2] + NFSR[15] + NFSR[8] \cdot NFSR[12] \\ &= b_2 + b_8 + b_8 b_{10} \end{aligned}$$

4.3.2.5 Metóda generate_keystream

Metóda načíta numerický keystream vygenerovaný skriptom Grain.py. Pomocou metódy `round_` vygenerujeme prednastavený počet polynómov keystreamu. K jednotlivým polynómom, generovanými počas výpočtu, pripočítame pravú stranu, respektíve numerický keystream. Takto vytvorenú sústavu polynómov uložíme do požadovaného súboru.

Ak používame algoritmus LSH, tak počítame stým, že spúšťame skript pre väčšie množstvo behov skriptu pre generovanie rovníc. Preto najprv načítame do teraz vygenerované polynómy sústavy. Do tejto načítanej sústavy následne pridáme nove vygenerované polynómy a túto celkovú sústavu znovu uložíme do prislúchajúceho súboru.

4.4 Skript MagmaSolver.py

Pomocou daného skriptu načítame sústavu polynómov, ktorú sme vygenerovali z predošlého skriptu. Potom vytvoríme spustiteľný kód, ktorý v magme nájde riešenie tejto sústavy pomocou Groebnerových báz.

4.4.1 Main

Skript spúšťame pomocou príkazu

```
python3 MagmaSolver.py bit_version number_of_rounds.
```

Parametre udávajú bitovú verziu šifry a celkový počet kôl. Vytvorí sa inštancia šifry a spustí sa výpočet na prelomenie šifry cez danú triedu. Po výpočte sa zapíše čas a využitá pamäť počas hľadania riešenia sústavy rovníc.

4.4.2 Trieda MagmaSolver

4.4.2.1 Konštruktor

Argumenty:

- `bit_version` - bitová verzia šifry
- `number_of_rounds` - počet kôl šifry

V konštruktore sa načíta daná vygenerovaná sústava rovníc do triednej premennej a uloží sa aj bitová verzia a počet kôl.

4.4.2.2 Metóda reduce_LSH

V tejto metóde sa nachádza implementácia algoritmu LSH pre zjednodušenie vygenerovanej sústavy polynómov. Popis tohto algoritmu môžeme nájsť v kapitole 5.4.1.

4.4.2.3 Metóda create_magma_script

Touto metódou sa vytvorí spustiteľný skript magmou potrebný na riešenie sústavy. Do skriptu sa zapíše taktiež kód pre vytvorenie booleovského polynomického okruhu s potrebnými premennými ako pri triede GrainPolynomial 4.3.2. Následne sa zapíšu jednotlivé polynómy do skriptu. Polynómy rozdelíme navrhnutou funkciou tak aby nenastala v magme chyba ”Segmentation fault”, ktorá je pravdepodobne spôsobená príliš veľkým množstvom sčítaných monómov v jednej premennej na jednom riadku. Preto vždy budeme jeden polynóm rozdeľovať do viacerých, kde jednotlivé premenné rozdelených polynómov majú vždy po max 10 000 monómov a následne tieto premenné sčítame do jedného výsledného monómu:

```
P0_0 := b0*b1*b14 + b0*b1 + b0*b2 + b0*b3*b14 + b0*b4*b15 + b0*b...;
P0_1 :=  b0*b1*b5 + b0*b1*b6 + b0*b1*b13 + b0*b1*b14*b15...;
P0 := P0_0 + P0_1;
P1_0 := b0*b3*b5 + b0*b3*b6 + b0*b3*b13 + b0*b3*b14*b15 + b0*b3*b14...;
P1 := P1_0;
```

Na príklade vyššie môžeme vidieť, že polynóm *P0* bol rozdelený na dva polynómy. Druhý polynóm *P1* mal menej ako 10 000 monómov, preto nebol rozdelený do viacerých polynómov. Následne sa pomocou funkcie *GroebnerBasis* dostupnej v Magme, vypočíta Groebnerova báza zadaných polynómov a následne pomocou taktiež dostupnej funkcie *Variety* sa vypočíta riešenie. Riešenie, respektíve riešenia sa zapíšu do výstupného súboru.

4.4.2.4 Metóda correctness_of_keys

Metóda načíta vygenerované riešenia z výstupného súbory Magmy. Riešenia spracuje a uloží do poľa kľúčov. Každý z načítaných kľúčov sa použije pre vytvorenie inštancie triedy Grain.py. Pomocou tejto inštancie sa vygeneruje nový keystream, ktorým sa potom zašifruje otvorený text a ten sa porovná s pôvodným šifrovaným textom. Týmto sa overí správnosť získaných kandidátov na kľúč.

Kapitola 5
Experimenty

Po popise implementácie prevodu danej šifry na sústavu polynomiálnych rovníc a hľadania jej riešenia sa budeme v tejto kapitole venovať experimentom. Cieľom týchto experimentov bude porovnať jednotlivé časy trvania daného prevodu šifry a časy pre hľadanie riešenia tejto sústavy rovníc pomocou Groebnerovych báz.

V tabuľkách sa budú nachádzať tieto skratky pre jednotlivé záznamy v riadkoch, respektíve stĺpcoch:

- PR - počet vygenerovaných polynomiálnych rovníc
- MIPM - priemerný minimálny počet monómov v jednom polynóme
- MAPM - priemerný maximálny počet monómov v jednom polynóme
- ČPŠ - priemerný čas prevodu šifry na sústavu polynomiálnych rovníc v **sekundách**
- PPŠ - priemerná použitá pamäť pri prevode šifry na sústavu polynomiálnych rovníc v **MB**
- ČVK - priemerný čas výpočtu kľúča v **sekundách**
- PVK - priemerná pamäť pri výpočte kľúča **MB**
- CV - priemerný čas celého výpočtu v **sekundách**
- CP - priemerná celková použitá pamäť v **MB**

5.1 Hardware

Spustenia jednotlivých experimentov prebiehali na stroji s nasledujúcimi špecifikáciami: operačný systém Ubuntu 20.04.4 LTS, dva procesory Intel Xeon Gold 6136, z ktorých každý obsahuje 12 jadier s frekvenciou o 3,0 GHz, 755GB pamäťe RAM.

5.2 16 bitová verzia šifry Grain

V tejto kapitole sme spustili testy na zmenšenú 16 bitovú verziu šifry Grain. Pozorovali sme priemerný čas a ďalšie hodnoty vypočítané z 25 spustení. Pri jednotlivých spusteniach sme náhodne vygenerovali kľúč aj inicializačný vektor nonce.

5.2.1 Rôzne hodnoty parametru r

Na pevno sme si zvolili parameter $n = 16$ a pozorovali sme ako sa budú meniť jednotlivé hodnoty pre rôzne veľkosti parametru r počas prevodu šifry na sústavu polynomiálnych rovníc a výpočte ich riešenia pomocou Groebnerových báz. Veľkosť keystreamu, respektíve počet vygenerovaných polynómov bol 16. Výsledky testov sú vidno v tabuľke 5.1.

Šifra(n,r)	PR	MIPM	MAPM	ČPŠ	PPŠ
Grain(16,8)	16	82	22242	1,94 ± 0,2	232 ± 4
Grain(16,16)	16	7106	32660	15,19 ± 2	272 ± 5
Grain(16,24)	16	32709	32753	44,8 ± 6,6	273 ± 6
Grain(16,32)	16	32735	32801	58.8 ± 2,8	274 ± 4,9
Grain(16,40)	16	32753	32757	69.3 ± 11.82	276 ± 5.45
Grain(16,48)	16	32796	32786	79.68 ± 1	277 ± 5,3
Grain(16,56)	16	32749	32782	90.35 ± 1	279 ± 5.5
Grain(16,64)	16	32801	32805	104.7 ± 1	280 ± 5

Šifra(n,r)	ČVK	PVK	CV	CP
Grain(16,8)	10,9 ± 1,14	157 ± 13	12,84	389
Grain(16,16)	448 ± 76,25	4751 ± 76	463,19	5023
Grain(16,24)	560.42 ± 165,41	18327 ± 3510	605,22	18600
Grain(16,32)	669,5 ± 164,5	21693 ± 4295	728,3	21967
Grain(16,40)	688.9 ± 141.6	23151 ± 3743	757,3	23427
Grain(16,48)	702.68 ± 158.8	22091 ± 5636	782,35	22368
Grain(16,56)	680.85 ± 134.5	23189 ± 4763	771,2	23465
Grain(16,64)	708.64 ± 132.5	22991 ± 5702	813,34	23268

Tabuľka 5.1 Spustenia pre 16 bitovú verziu šifry Grain pri rôznych hodnotách parametru k

Z tabuľky 5.1 sme zistili nasledujúce výsledky. Pri zvyšovaní počtu kôl sa nám zvyšoval úmerne čas a pamäť pri prevode šifry na polynomiálnu sústavu rovníc. Pri hľadaní riešenia vygenerovanej sústavy polynomiálnych rovníc je vidno, že čas taktiež rástol pri zvyšujúcom sa počte kôl, respektíve zvyšujúcom sa parametri r. Ale pri parametri $48 \leq r \leq 64$ sa čas veľmi nezvyšoval a bol v rovnakom rozmedzí. Pri zmene parametra r z 8 na 16 môžeme vidieť väčší nárast času.

Majme nasledujúce vlastnosti zmenšenej verzie 16 bitovej verzie šifry Grain.

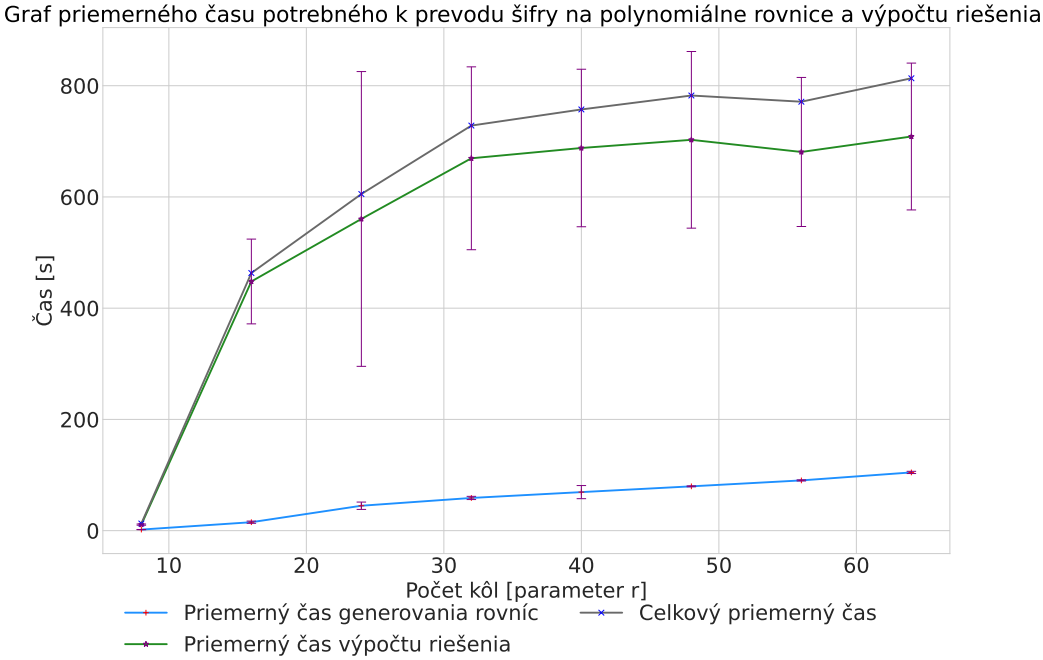
$$\begin{aligned} y &= NFSR[2] + NFSR[15] + NFSR[8] \cdot NFSR[12] \\ next\_bit\_lfsr &= LFSR[0] + LFSR[7] \\ next\_bit\_nfsr &= NFSR[0] + NFSR[3] + NFSR[11] \cdot NFSR[13] \end{aligned}$$

V registroch sa pri 8 kolách nestihne premiešať dostatok premenných. Jednotlivé nové polynómy nebudú tak zložité, pretože na indexoch 0,2,3,8 stále budú len jednoduché polynómy s jedným monómom typu b_x, $0 \leq x \leq 15$. Preto sa už pri 16 kolách zvyšuje čas prevodu šifry na sústavu rovníc. Tam sa totiž na dané indexy dostanú už zložitejšie polynómy. Taktiež sa zvyšovali aj počty monómov v jednotlivých polynómoch keystreamu. Môžeme vidieť podobný nárast minimálneho počtu aj maximálneho počtu monómov v jednom polynóme. Ide o podobný princíp ako pri zvyšujúcom sa čase. Od 24 kôl sa oboje počty držia v rozmedzí od 32709 do 32805 monómov v jednom polynóme.

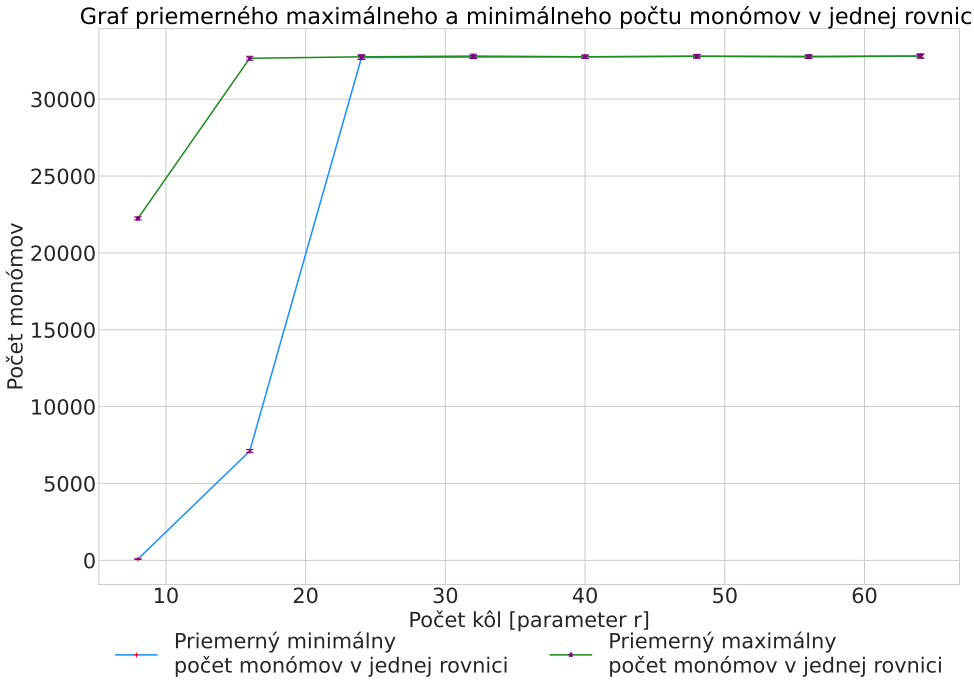
Jednotlivé vyššie popísané výsledky vyplývajúce z tabuľky 5.1 sú potvrdené aj na nasledujúcich grafoch. Na grafe 5.1 môžeme vidieť ako sa vyvíjal čas prevodu šifry na rovnice a

výpočet riešenia pomocou Magmy. Na ďalšom grafe 5.2 je vidno priemerné počty monómov v jednotlivých polynómoch keystreamu.

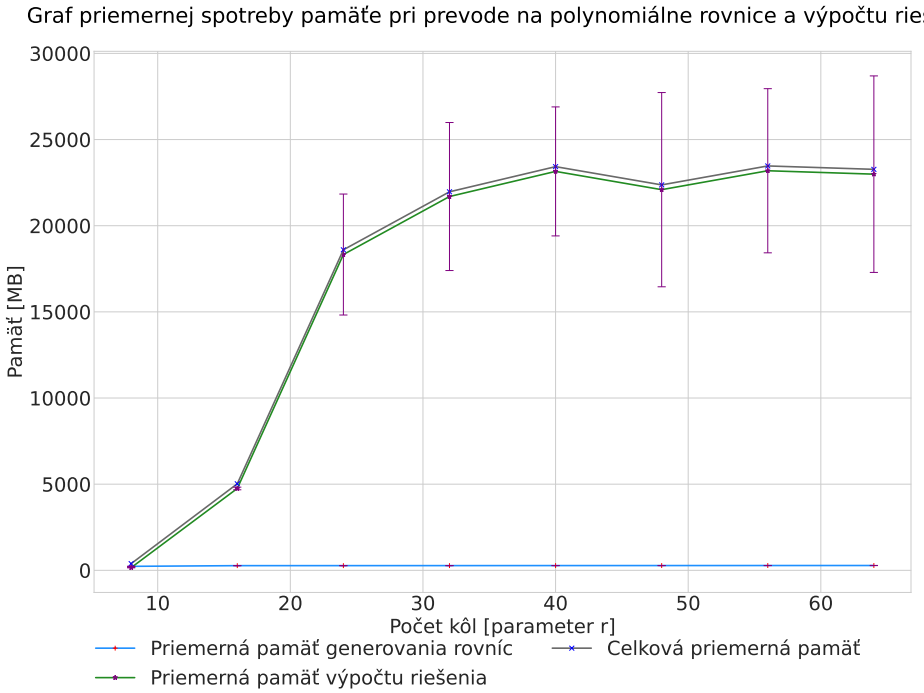
Na poslednom grafe 5.3 vidieť priemernú spotrebovanú pamäť pri prevode šifry na rovnice a výpočtu riešenia. Pamäť pri generovaní rovníc je veľmi malá a skoro nemenná. Zatiaľ čo pamäť pri výpočte Groebnerových báz a hľadaní riešenia sa zvyšovala podobne ako pri sledovaní priemerného času v grafe 5.1



Obr. 5.1 Graf priemerného času potrebného k prevodu šifry na polynomiálne rovnice a výpočet riešenia pre Grain(16,k)



Obr. 5.2 Graf priemerného maximálneho a minimálneho počtu monómov v jednom polynóme pre Grain(16,k)



Obr. 5.3 Graf priemernej pamäte potrebnej k prevodu šifry na polynomiálne rovnice a výpočet riešenia pre Grain(16,k)

5.2.2 Rôzne veľkosti keystreamu pre Grain(16,64)

Pozrieme sa na výsledky testov pozorovania času so zvyšujúcim sa počtom polynómov v keystreame. Testy sme spustili pre zmenšenú 16 bitovú verziu šifry Grain s parametrami $n = 16$ a $r = 64$ vypočítaným v kapitole 4.2.

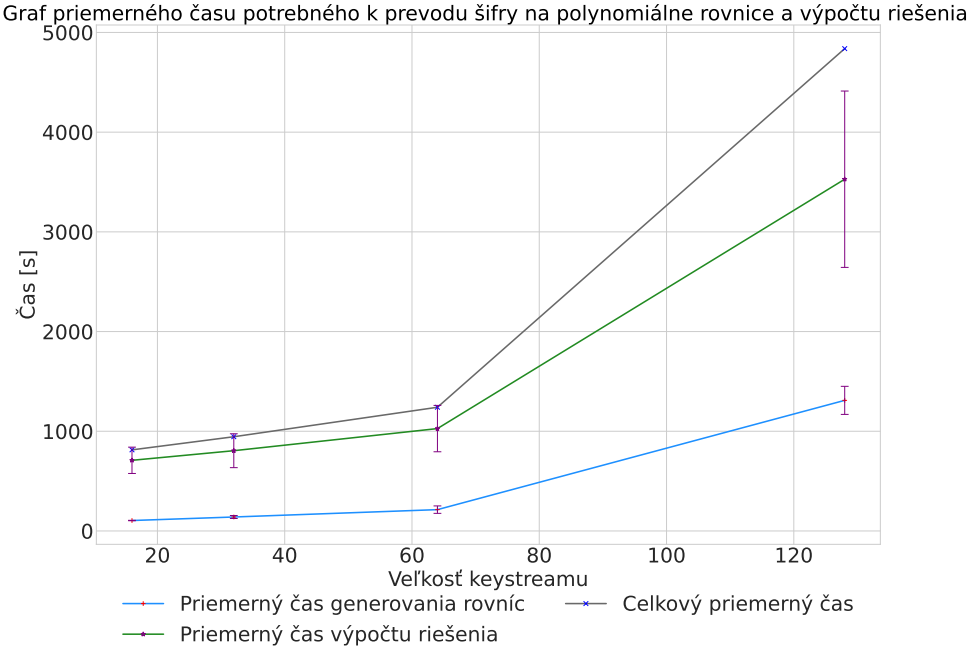
Šifra(n,r)	PR	MIPM	MAPM	ČPŠ	PPŠ
Grain(16,64)	16	32801	32805	104.7 ± 1	280 ± 5
Grain(16,64)	32	32660	32701	140 ± 15	272 ± 5
Grain(16,64)	64	32747	32758	213.9 ± 37,12	286 ± 5
Grain(16,64)	128	32713	32745	1309,9 ± 140.77	349 ± 3

Šifra(n,r)	ČVK	PVK	CV	CP
Grain(16,64)	708.64 ± 132.5	22991 ± 5702	813,34	23268
Grain(16,64)	805 ± 170	22567 ± 4042	945	5023
Grain(16,64)	1026,8 ± 232,46	21724 ± 3942	1240,7	18600
Grain(16,64)	3528 ± 884,42	22486.67 ± 3945	4837,9	21967

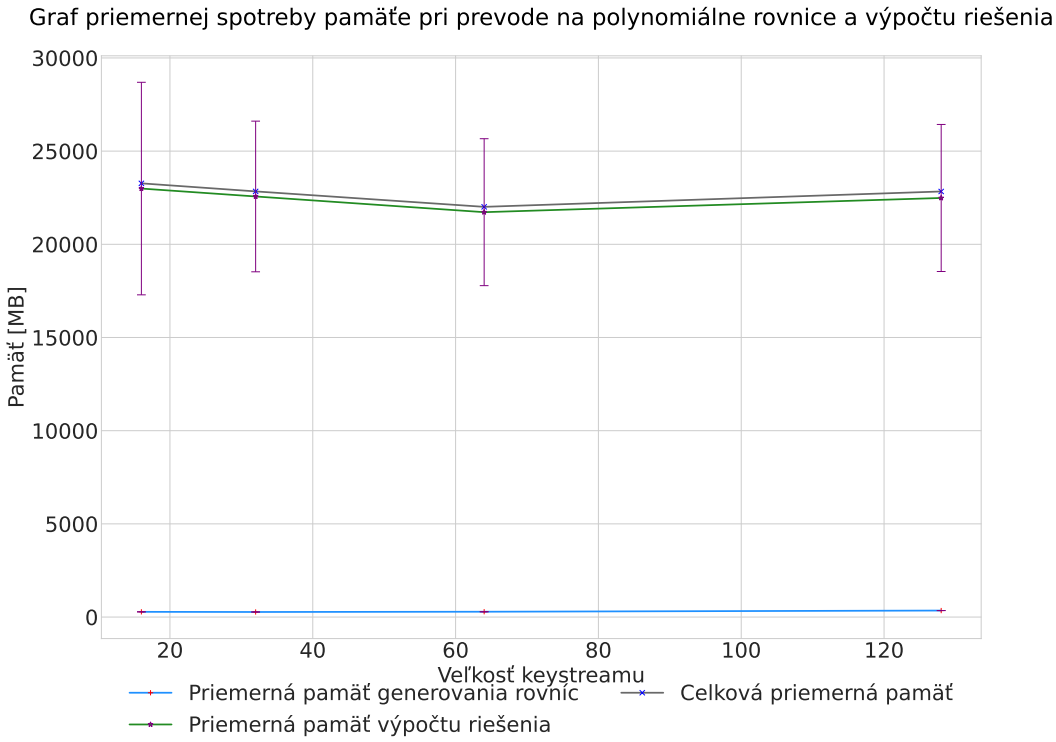
Tabuľka 5.2 Rôzne veľkosti keystreamu pre šifru Grain(16,64)

V tabuľke 5.2 môžeme vidieť, že čas generovania rovníc rástol so zväčšujúcim sa keystreamom rovnako ako keď sme zvyšovali počet kôl. Jednotlivé počty monómov sa už nezvyšujú ani nezmenšujú. Predpokladáme, že dané polynómy sú si navzájom podobné len stým, že sa vždy odstráni niektorý monóm a pridá sa ďalší. Takto sa v ďalších rovniciach striedajú podobné monómy a preto pri väčšom keystreame trvá výpočet v magme dlhšie a dlhšie, keďže jednotlivé polynómy nepridávajú žiadnu informáciu do výsledného keystreamu.

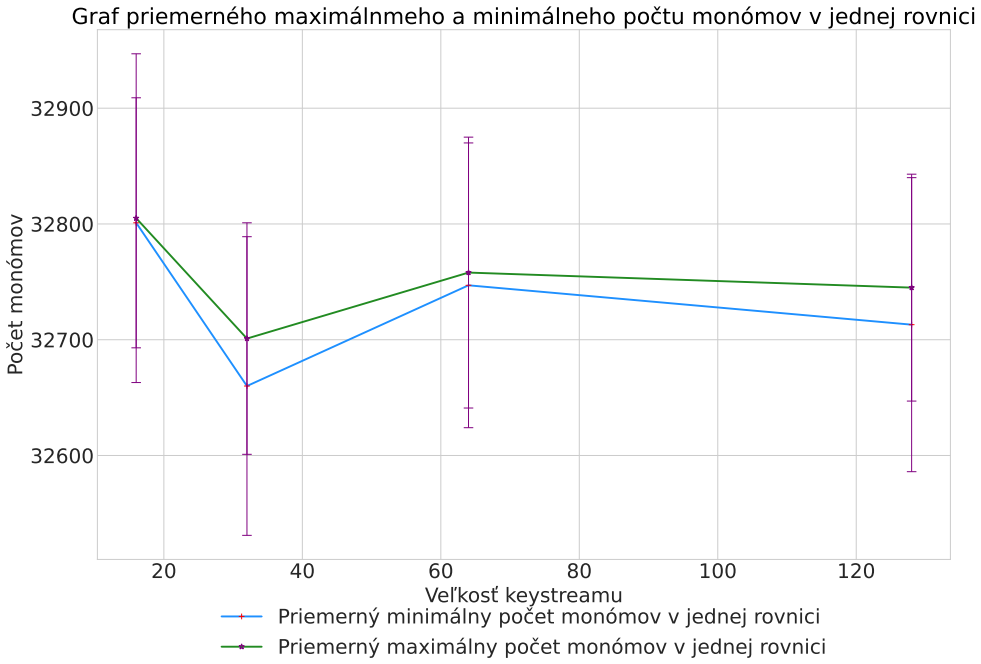
Na ďalších grafoch nižšie sú tieto tvrdenia lepšie znázornené. Z grafu 5.4 vidíme ako sa vyvíjal čas prevodu šifry na sústavu polynomiálnych rovníc a výsledný čas riešenia tejto sústavy.



Obr. 5.4 Graf priemerného času potrebného k prevodu šifry na polynomiálne rovnice a výpočet riešenia pre Grain(16,64) pri zvyšovaní veľkosti keystreamu



Obr. 5.5 Graf priemernej pamäte pre Grain(16,64) pri zvyšovaní veľkosti keystreamu



Obr. 5.6 Graf priemerného maximálneho a minimálneho počtu monómov v jednom polynóme pre Grain(16,64) pri zvyšovaní veľkosti keystreamu

Na grafe 5.5 môžeme vidieť zobrazenie potrebnej priemernej pamäte pre Grain(16,64) zaznamenanej pri experimentoch pri zvyšovaní veľkosti keystreamu. Pre rôzne veľkosti bola použitá pamäť približne rovnako veľká. Na grafe 5.6 môžeme vidieť ako rástli počty monómov pri zvyšovaní veľkosti keystreamu pre 16 bitovú verziu šifry.

5.3 Ďalšie zmenšené verzie šifry Grain

Počas generovania rovníc zmenšených verzii šifry Grain sme narazili na problémy s výpočtovým časom. V tejto kapitole sa pozrieme na výsledky pre niektoré ďalšie zmenšené verzie šifry, ktoré sa nám podarilo otestovať.

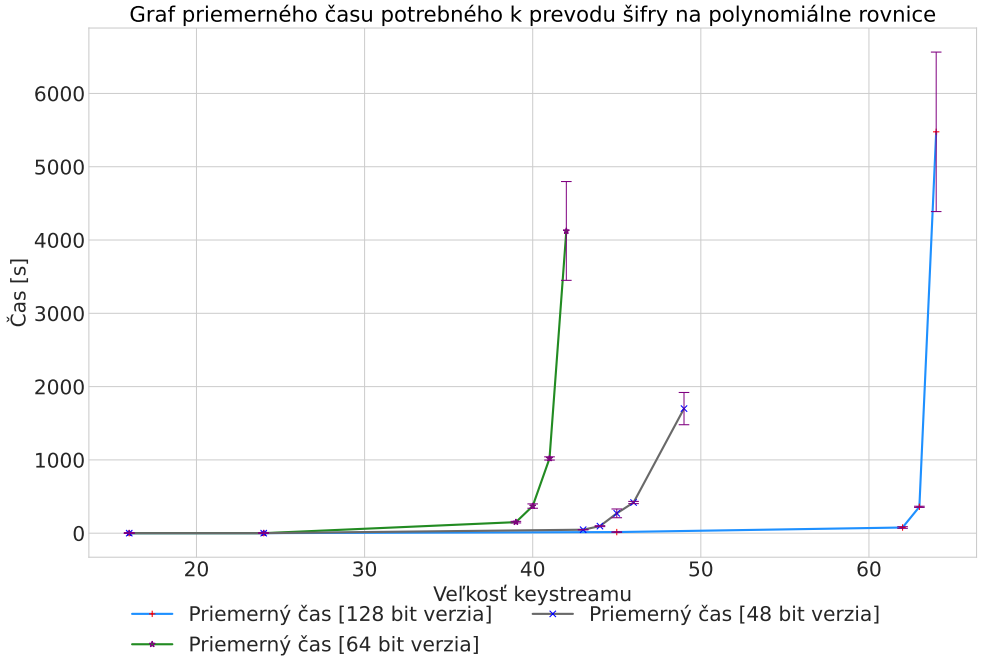
Šifra(n,r)	PR	MIPM	MAPM	ČPŠ	PPŠ
Grain(24,8)	16	5	1480	11,4 ± 2	256 ± 4
Grain(24,8)	24	5	77024	5822,75 ± 804	388 ± 10
Grain(32,8)	16	2	25.5	4665,2 ± 1131,7	689 ± 64
Grain(48,8)	16	25	48	0,62 ± 0,1	205 ± 0,3
Grain(48,8)	24	34	3722	0,62 ± 0,1	206 ± 0,2
Grain(48,8)	43	25	123533	47,7 ± 2,3	264 ± 3
Grain(48,8)	44	25	211724	96,3 ± 6,3	302 ± 5
Grain(48,8)	45	25	1254065	270,7 ± 59,7	383 ± 16
Grain(48,8)	46	23	4921432	418,7 ± 10	434 ± 7
Grain(48,8)	48	25	16520340	1622,37 ± 220	1099 ± 42
Grain(64,8)	16	5	29	0,62 ±0,09	206 ± 0,2
Grain(64,8)	24	4	176	0,6 ± 0,08	208 ± 0,3
Grain(64,8)	39	5	3897	150,8 ± 10,6	623 ± 3
Grain(64,8)	40	5	8247	370 ± 29,6	1096 ± 35
Grain(64,8)	41	5	13178	1020,5 ± 22	1899 ± 76
Grain(64,8)	42	5	23810	4124,98 ± 674	6065 ± 447
Grain(128,8)	16	8	9	0,63 ± 0,15	206 ± 0,3
Grain(128,8)	24	9	9	1,2 ± 0,4	206 ± 0,35
Grain(128,8)	45	9	36	15,3 ± 2,5	210 ± 0,3
Grain(128,8)	62	8	73766	77,65 ± 11	337 ± 13
Grain(128,8)	63	8	69218	359,8 ± 15,8	544 ± 22
Grain(128,8)	64	9	101076	5476,3 ± 1088	2339 ± 284

Tabuľka 5.3 Časy pre ďalšie zmenšené verzie šifry Grain

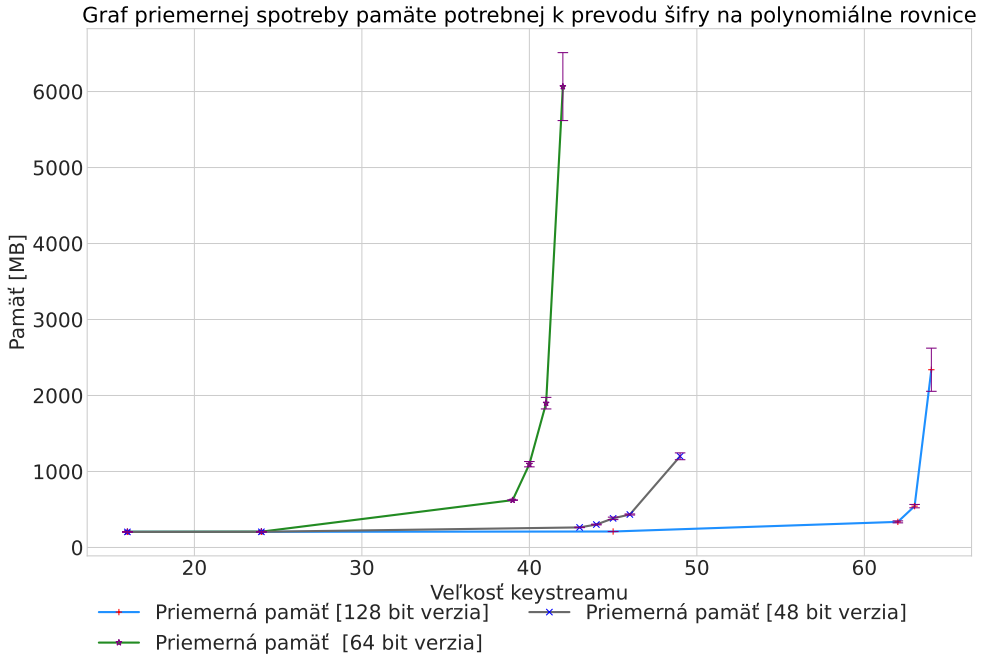
V tabuľke 5.3 môžeme vidieť výsledky vykonané pre rôzne ďalšie zmenšené varianty šifry Grain. Experimenty sme vykonali pre rôzne bitové veľkosti a pre fixný parameter $r = 8$. Cieľom bolo vygenerovať počet rovníc rovný bitovej veľkosti. Pre rôzne typy verzii sa nám podarilo vygenerovať vždy rôzny počet rovníc keystreamu a ani pri jednej sa nepodarilo dosiahnuť veľkosti bitovej verzii. Pre väčší počet kôl sa nám z vysokého výpočtového času nepodarilo vykonať experimenty. Preto sme nepoužili viac než 8 kôl v jednotlivých verziách. Veľkosť keystreamu sme sa vždy pokúsili vygenerovať maximálnu možnú. Napríklad pri 64 bitovej verzii sme dosiahli maximum veľkosti 42. Pri 32 bitovej bolo maximum 16.

Nasledujúci rozbor časov platí pre všetky verzie. Preto sa pozrieme len na časy pre 128 bitovú verziu šifry Grain. Prvé tri indexy ku ktorým pristupujeme v NFSR v jednotlivých častiach šifry sú 95, 94 a 93. Takže ak máme 8 kôl a k tomu veľkosť keystreamu 16, dostaneme celkový počet kôl rovný 24. V tom prípade sa počas inicializácie a generovania keystreamu nedostanú na používané indexy zložitejšie polynómy. Polynómy budú vyzerať ako jeden jednoduchý monónom, napr. b_x.

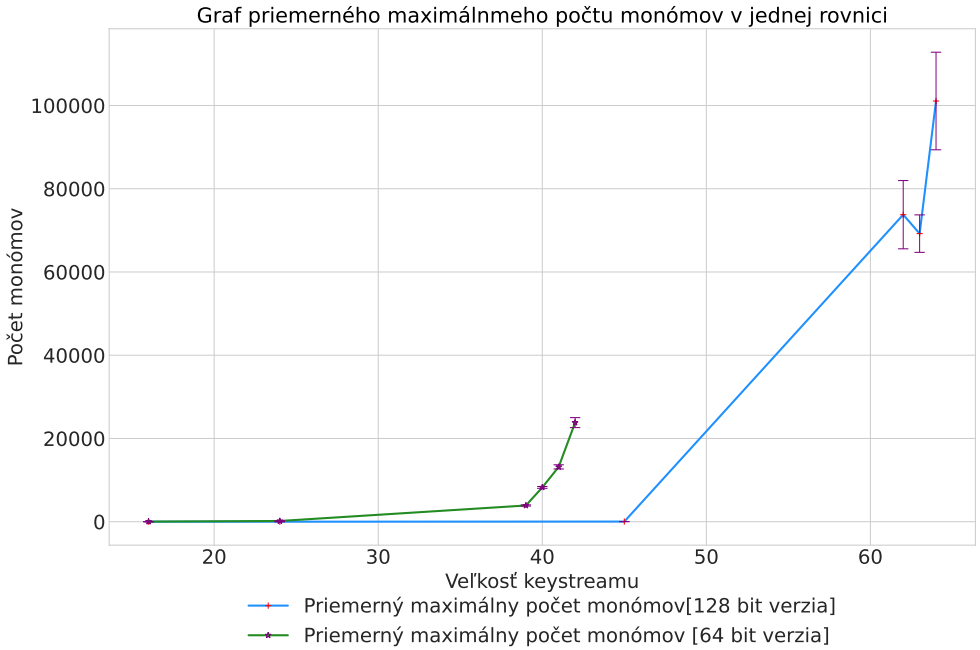
Podobne to bude až po veľkosť 24. Pri veľkosti 45 a 64 vidíme, že generovanie trvalo dlhšie. Maximálny počet monómov sa zväčšil z 9 na 36, čo potvrdzuje vyššie závery ohľadom indexov.



Obr. 5.7 Graf priemerného času pre niektoré ďalšie verzie šifry



Obr. 5.8 Graf priemernej pamäte pre niektoré ďalšie verzie šifry



Obr. 5.9 Graf priemerného maximálneho počtu monómov v jednom polynóme pre niektoré ďalšie verzie šifry

Rozhodli sme sa na grafoch 5.7,5.8 a 5.9 zobraziť čas, pamäť a maximálny počet monómov generovania rovníc pre niektoré zmenšené verzie šifry s počtom kôl 8.

5.3.1 Výpočtová zložitosť jednotlivých častí šifry

Rozhodli sme sa pre 128 bitovú verziu šifry s počtom kôl 8 a dĺžkou keystreamu 63 zobraziť tabuľku s jednotlivými časmi častí šifry.

Šifra(n,r)	Grain(128,8)
PR	63
Celkový čas	514.4
NFSR	7,22 ± 50
LFSR	1,54 ± 1
Funkcia y	0,012 ± 0,08
NFSR posledné 71. kolo	424,55
LFSR posledné 71. kolo	3,91
Y posledné 71. kolo	0,08
NFSR 70. kolo	85,7
LFSR 70. kolo	5,2
Y 70. kolo	0,75
NFSR 69. kolo	0,7
LFSR 69. kolo	2,2
Funkcia y 69. kolo	0,75

Tabuľka 5.4 Časy pre jednotlivé časti šifry Grain

Z tabuľky 5.4 sme zistili priemerné časy jednotlivých častí šifry, ktoré sa nachádzajú na riadkoch 4, 5 a 6. Na ďalších riadkoch môžeme vidieť trvania častí šifry v daných kolách. Môžeme vidieť, že v poslednom kole trvalo NFSR až 424 sekúnd, čo je skoro celkový čas spustenia generovania rovníc. Keďže len jedna časť trvá väčšinu času skriptu tak nám paralelizované spustenie týchto troch častí nepomôže. Jeden zo spôsobov ako urýchliť výpočet je paralelizovať násobenie polynómov, čo je nad rámec tejto práce.

5.4 Redukcia počtu vygenerovaných polynómov v sústave pomocou algoritmu LSH

Na základe výsledkov z diplomovej práci Ing. Jany Beruškovej sme sa rozhodli použiť algoritmus Locality Sensitive Hashing, skrátene LSH [22], na zjednodušenie vygenerovanej sústavy polynómov. Tým sme sa snažili zrýchliť výpočet Groebnerových báz v Magme. Algoritmus využijeme z implementácie, ktorú vytvorila Ing. Jana Berušková vo svojej diplomovej práci [23]. Princíp spracovania polynómov je založený na probléme hľadania najbližšieho suseda, respektíve približne najbližšieho suseda [24].

▶ **Definícia 5.1.** *Majme prvok p. Cieľom je nájsť také* $x \in \{x_1, \ldots, x_n\}$*, pre ktoré je vzdialenosť d(p,x) najmenšia pre všetky možné x.*

Pri použití algoritmu LSH bude problém definovaný týmto spôsobom. Najskôr spočítame XOR dvoch polynómov a uložíme si výsledný polynóm. Vzdialenosť d definujeme ako počet monómov výsledného polynómu. Najpodobnejšie polynómy potom zoradíme podľa vzdialeností od najmenšej po najväčšiu. Z nich vytvoríme novú sústavu tvorenú jednoduchšími polynómami pomocou XORu najpodobnejších polynómov. Táto sústava sa použije pre výpočet Grobnerovej bázy. V ďalšej podkapitole sa pozrieme podrobnejšie na implementáciu algoritmu LSH.

5.4.1 Popis implementácie algoritmu LSH

LSH [22] je celosvetovo populárna technika používaná pri hľadaní približne najbližšieho suseda [24]. Používa sa napríklad pri vyhľadávaní pomocou podobných výrazov v internetovom vyhľadávači alebo pre odporúčanie nových produktov na základe porovnania podobností histórie nákupov zákazníkov.

Vysvetlili sme si podmienku ako nájsť najpodobnejšie polynómy z nejakej množiny polynómov. Podmienka pre hľadanie najpodobnejších polynómov nám nestačí. Museli by sme porovnať každý polynóm s každým. To by sme pracovali so zložitosťou $O(n \cdot log(n))$. Preto potrebujeme porovnávať len tie polynómy, ktoré považujeme za vhodné páry kandidátov na najpodobnejšie polynómy. Na výber kandidátov slúži práve algoritmus LSH a my sa pozrieme na to ako je daný algoritmus definovaný.

V princípe LSH niekoľko krát rozdeľuje a hashuje jednu rovnakú vzorku. Následne dokážeme porovnať, či bol nejaký pár vektorov rovnako zahashovaný a tak dostaneme jeden pár, respektíve kandidáta. LSH si rozdelíme do troch základných krokov a to Shingling, MinHashing a LSH.

5.4.1.1 Krok Shingling

V našom prípade budeme vytvárať pre každý polynóm množinu, ktorá reprezentuje všetky jeho monómy. Tieto množiny vytvoríme pre každý polynóm našej sústavy. Zoberieme všetky získané monómy a vytvoríme z nich jednu spoločnú množinu bez duplikátov. Táto množina bude reprezentovať všetky monómy, ktoré sa nachádzajú v našej sústave. Z tejto množiny vytvoríme slovník, kde každý monóm bude mať svoje poradové číslo.

▶ **Príklad 5.2.** Majme sústavu o dvoch polynómoch $f = x_1 + x_2 + x_2x_3 + x_4$ a $g = x_1 + x_2 + x_2x_4 + x_3$. Množiny monómov sú:

$$m_1 = \{x_1, x_2, x_2x_3, x_4\}$$

$$m_2 = \{x_1, x_2, x_2x_4, x_3\}$$

Zo spoločnej množiny obsahujúcej monómy celej sústavy vytvoríme slovník:

$$D = \{x_1 : 1, x_2 : 2, x_2x_3 : 3, x_2x_4 : 4, x_3 : 5, x_4 : 6\}$$

Pomocou slovníku vytvoríme *riedke binárne vektory*, ktoré budú reprezentovať monómy obsiahle v jednotlivých polynómoch. Najskôr vytvoríme prázdny vektor plný núl, ktorého veľkosť bude rovná veľkosti slovníku. Na každé miesto s indexom poradového čísla monómu obsiahleho v polynóme priradíme jednotku. Poradové číslo získame zo slovníku.

▶ **Príklad 5.3.** Pokračujeme v spracovaní sústavy o dvoch polynómoch f a g. Zoberme si vytvorené množiny m_1, m_2 a slovník D. Riedke binárne vektory v_1 a v_2 sú:

m_1	m_2	D	v_1	v_2
x_1,	x_1,	$x_1 : 1$,	1	1
x_2,	x_2,	$x_2 : 2$,	1	1
x_2x_3,	x_2x_4,	$x_2x_3 : 3$	1	0
x_4,	x_3,	$x_2x_4 : 4$,	0	1
		$x_3 : 5$,	0	1
		$x_4 : 6$	1	0

5.4.1.2 Krok MinHashing

V tomto kroku budeme používať hashovanie na vytvorenie hustých vektorov z tých riedkych. Tieto husté vektory budeme označovať *signatúry*. Pre každú pozíciu v signatúre vygenerujeme náhodnú permutáciu o veľkosti slovníku. Veľkosť množiny *MinHash* obsahujúcej permutácie bude rovnaká ako veľkosť signatúry. Túto náhodnú permutáciu budeme prechádzať a hľadať najskôr číslo jedna. Zistíme na akej pozícii sa nachádza a skontrolujeme, či na rovnakej pozícii v riedkom vektore je číslo jedna. Ak áno, tak uložíme číslo jedna ako prvý prvok signatúry. Ak nie, tak budeme opakovať tento postup pre ďalšie čísla v poradí až kým nenájdeme číslo, ktoré sa nachádza na pozícii, kde je v riedkom vektore jednotka.

▶ **Príklad 5.4.** Majme napríklad riedky vektor v_1 a uvažujme množinu *MinHash* obsahujúcu permutácie. Veľkosť jednej signatúry je rovná veľkosti množiny *MinHash*.

$MinHash$	v_1
| 3 | 2 | 6 | 6 |	**1**
| 4 | **1** | **3** | 2 |	**1**
| **2** | 3 | 5 | **1** |	**1**
| 5 | 6 | 1 | 4 |	0
| 1 | 4 | 2 | 5 |	0
| 6 | 5 | 4 | 3 |	**1**

Jednotka v prvej permutácii sa nachádza na mieste, kde je vo vektore v_1 prvok 0. Preto prejdeme na dvojku v prvej permutácii. Tá sa nachádza na mieste v permutácii, kde je na rovnakom mieste vo vektore v_1 uložený prvok 1. Preto bude na prvom mieste v signatúre s_1 číslo 2. Teraz prejdeme na druhú permutáciu. Nájdeme v nej číslo 1 a skontrolujeme, či sa na rovnakom mieste vo vektore v_1 nachádza taktiež číslo 1. Keďže na tomto mieste vo vektore v_1 je číslo 1, tak na ďalšie miesto v signatúre zapíšeme 1. V tomto kroku máme signatúru $s_1 = \{2, 1\}$. V ďalšej permutácii budeme prechádzať čísla až po 3, pretože na mieste, kde sa nachádza číslo 3 je vo vektore v_1 uložená 1. Z poslednej permutácie dostaneme číslo 1. Týmto postupom dostaneme signatúru $s_1 = \{2, 1, 3, 1\}$ pre polynóm f a signatúru $s_2 = \{1, 1, 1, 2\}$ pre polynóm g.

5.4.1.3 Krok LSH

Ak by sme ďalej pracovali s celými signatúrami znamenalo by to, že pre rovnaké polynómy by sa vygeneroval rovnaký binárny vektor a taktiež aj signatúra. Tým pádom by sme hľadali rovnaké polynómy a mohli by sme nájsť menej kandidátov. Preto namiesto spracovania celej signatúry prostredníctvom hashovacej funkcie spracujeme iba jej pod časti. Rozdelením signatúry získame viac možností na identifikáciu podobných polynómov. Každú časť spracujeme pomocou rovnakej hash funkcie a uložíme do množiny *buckets*. Stačí nám nájsť dve rovnaké časti a označíme celé dva polynómy, ktorých dve časti sme porovnávali, ako pár kandidátov.

▶ **Príklad 5.5.** Signatúry $s_1 = \{2, 1, 3, 1\}$ a $s_2 = \{1, 1, 1, 2\}$ rozdelíme na 4 časti. Takže budeme brať vždy jeden prvok zo signatúry. Množinu slovníkov si označíme buckets. Táto množina bude mať veľkosť odpovedajúcu počtu častí signatúr. Budeme indexovať cez jednotlivé časti signatúr. Tieto časti použijeme ako kľúč do slovníku nachádzajúcom sa na indexe podľa poradia časti. Príslušná hodnota pre kľúč bude pole, do ktorého pridáme poradové číslo polynómu, ktorému odpovedá signatúra (prvá signatúra $\rightarrow$ 1, druhá $\rightarrow$ 2).

Množina buckets bude po indexovaní cez prvú signatúru obsahovať:

Buckets

$(2:1)$
$(1:1)$
$(3:1)$
$(1:1)$

V riadkoch sa nachádzajú jednotlivé pod časti a poradie polynómu. Po spracovaní druhej signatúry sa množina zmení nasledovne:

Buckets

$(2:1,1:2)$
$(1:1,2)$
$(3:1,1:2)$
$(1:1,2:2)$

Pridali sme časti ďalšej signatúry do množiny. Vidíme, že ak sa v danom riadku časť ešte nenachádza, tak pre ňu vytvoríme nový záznam. Pri druhom riadku sme pridali poradové číslo polynómu do poľa už vytvoreného záznamu, kvôli zhodnosti častí signatúr. Tieto dva polynómy tvoria pár kandidátov.

Posledným krokom je skontrolovať každé pole v slovníkoch. Ak má toto pole veľkosť väčšiu ako 1 tak vytvoríme všetky možné kombinácie poradových čísel polynómov, ktoré sa nachádzajú v tomto poli. Tak dostaneme kandidátov na podobné polynómy. Pre každý pár spočítame ich vzdialenosť, ktorú sme si definovali na začiatku popisu algoritmu. Vzniknuté polynómy zoradíme podľa vzdialeností od najmenšej po najväčšiu a požadovaný počet použijeme pre výpočet Groebnerových báz.

5.4.2 Redukcia počtu polynómov 16 bitovej verzie šifry Grain

Najskôr sme LSH použili na 16 bitovú verziu šifry Grain s počtom kôl 16. Na prvom riadku tabuľky 5.5 sú výsledky experimentov pre verziu šifry pred použitím LSH s veľkosťou keystreamu 16. Pre viacero známych nonce sme spustili generovanie sústavy rovníc s počtom polynómov 16. Z týchto jednotlivých sústav sme vytvorili jednu veľkú sústavu a na tú sme aplikovali redukciu pomocou LSH. Vybrali sme buď 16, 24 alebo 32 najlepších párov a z nich sme vytvorili nové jednoduchšie sústavy. Tieto nové sústavy sme použili pre výpočet Groebnerovych báz.

V tabuľke 5.5 sme pridali nový stĺpec pre celkový čas skriptu MagmaSolver.py a to celkový čas výpočtu kľúča, skrátene CČVK. Ten reprezentuje ako čas generovania a riešenia Groebnerovych báz, tak aj prípravu dát a redukciu pomocou LSH. Taktiež sme pridali nový stĺpec pre počet nonce, skrátene PN. Vynechali sme niektoré výsledky pre generovanie rovníc. Keďže napríklad výpočet rovníc trval okolo 16 sekúnd a tento výpočet sme spúšťali toľko krát koľko bol počet nonce. Výsledky pre generovanie rovníc môžeme nájsť v kapitole 5.2.1.

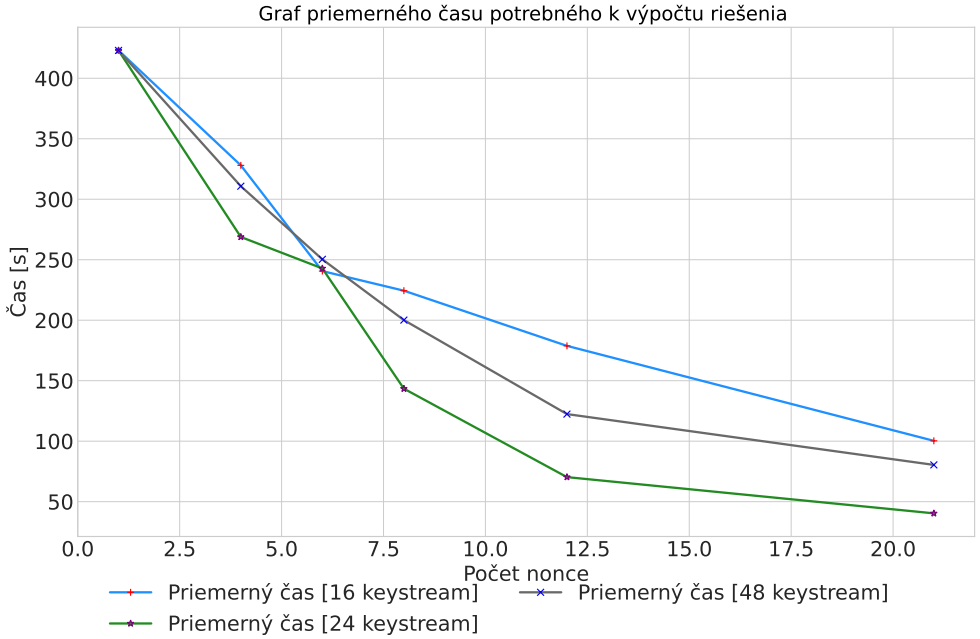
Na grafe 5.10 môžeme vidieť čas výpočtu Groebnerových báz a hľadania riešenia pre 3 behy experimentov. Čas výpočtu v Magme klesal so zvyšujúcim sa počtom nonce a použitím algoritmu LSH pre zjednodušenie sústavy polynómov. Pre viac možností výberu kandidátov z väčšej sústavy polynómov sme vždy dostali lepší čas riešenia. Pri výbere väčšieho počtu polynómov z celkovej sústavy než bola veľkosť kľúča, napríklad 24, sme dosiahli rýchlejšieho času riešenia. Zistili sme, že efektívnosť klesala pri použití 48 polynómov pre výpočet riešenia oproti použití 24 polynómov. Preto je lepšie pre danú variantu šifry použiť menší počet polynómov než je 48 ale zároveň väčší než je veľkosť kľúča.

Šifra(n,r)	PR	PN	MIPM	MAPM	ČVK	CČVK
Grain(16,16)	16	1	7189	32729	423,31	425
Grain(16,16)	16	4	4968	14231	328,4	1162,4
Grain(16,16)	16	6	4691	11411	240,6	2465,48
Grain(16,16)	16	8	4883	7100	224,4	4045,3
Grain(16,16)	16	12	4448	6134	178,8	8611,9
Grain(16,16)	16	21	3976	5084	100,3	25448,6
Grain(16,16)	24	4	4865	23880	268,86	1050
Grain(16,16)	24	6	4565	14168	242,8	2390,20
Grain(16,16)	24	8	4883	8000	143,4	3915,3
Grain(16,16)	24	12	4365	6964	70,3	8496,10
Grain(16,16)	24	21	4026	6024	40,35	25248.66
Grain(16,16)	48	4	4795	27353	310,76	1249,2
Grain(16,16)	48	6	3847	19176	250,31	2406,7
Grain(16,16)	48	8	3891	9500	200,1	4002,7
Grain(16,16)	48	12	3847	8444	122.4	8532
Grain(16,16)	48	21	3850	7234	80.4	25323

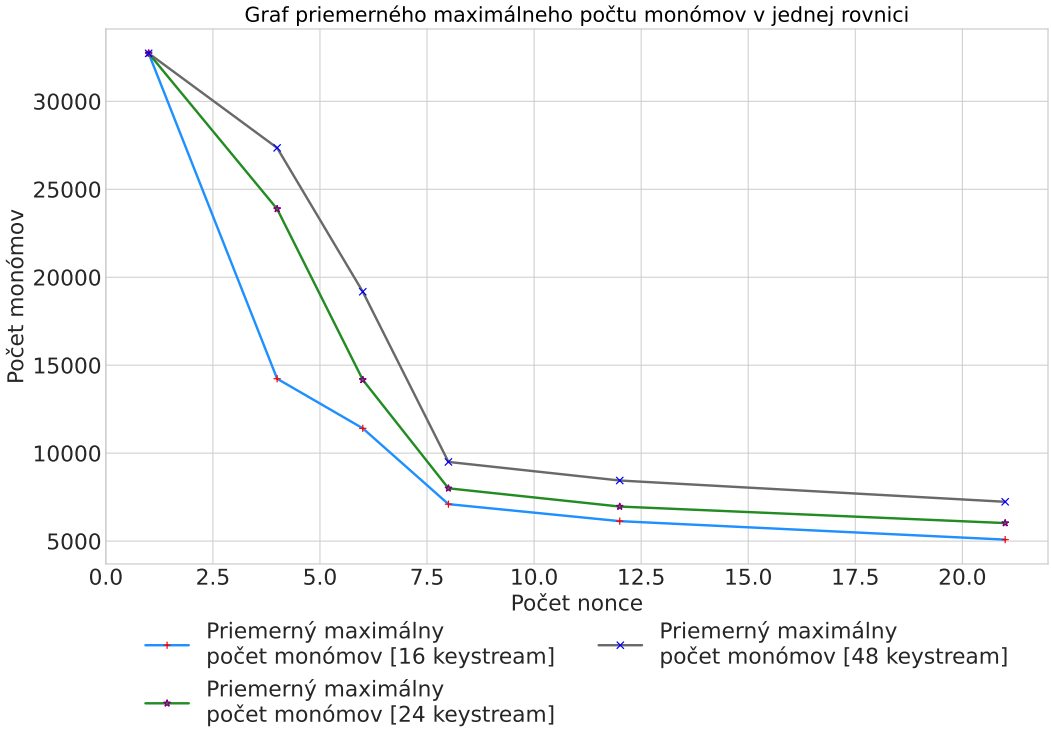
Tabuľka 5.5 Redukcia počtu polynómov pre Grain(16,16)

Zároveň si ale musíme uvedomiť, že čas celkovej réžie skriptu s použitím LSH sa zvyšoval. Je to spôsobené dôsledkom hľadania kandidátov na jednoduchšie polynómy z väčšej sústavy polynómov.

Keďže priemerný počet minimálneho počtu monómov nebol medzi riešeniami príliš rozdielny, rozhodli sme sa na grafe 5.11 zobraziť maximálny počet monómov v polynómoch. Maximálny počet monómov klesal so zvyšujúcim sa počtom nonce. Napríklad pri počte nonce 21 klesol maximálny počet monómov pôvodnej sústavy z 32729 až na 5084. Z toho vyplýva, že sa nám podarilo vytvoriť novú jednoduchšiu sústavu polynómov z tej pôvodnej použitím algoritmu LSH. Môžeme taktiež vidieť rozdiel priemerného maximálneho počtu monómov medzi 3 behmi experimentov. Keďže máme zoradených kandidátov podľa vzdialeností, tak ďalšie pridané polynómy do sústavy pre výpočet Groebnerových báz majú väčší počet monómov. Preto môžeme vidieť, že pri zvyšujúcom sa počte jednoduchších polynómov vo výslednej sústave, ktorá sa použije pre výpočet riešenia, je priemerný maximálny počet monómov vyšší.



Obr. 5.10 Graf priemerného času pre Grain(16,16) pri použití LSH



Obr. 5.11 Graf priemerných počtov monómov pre Grain(16,16) pri použití LSH

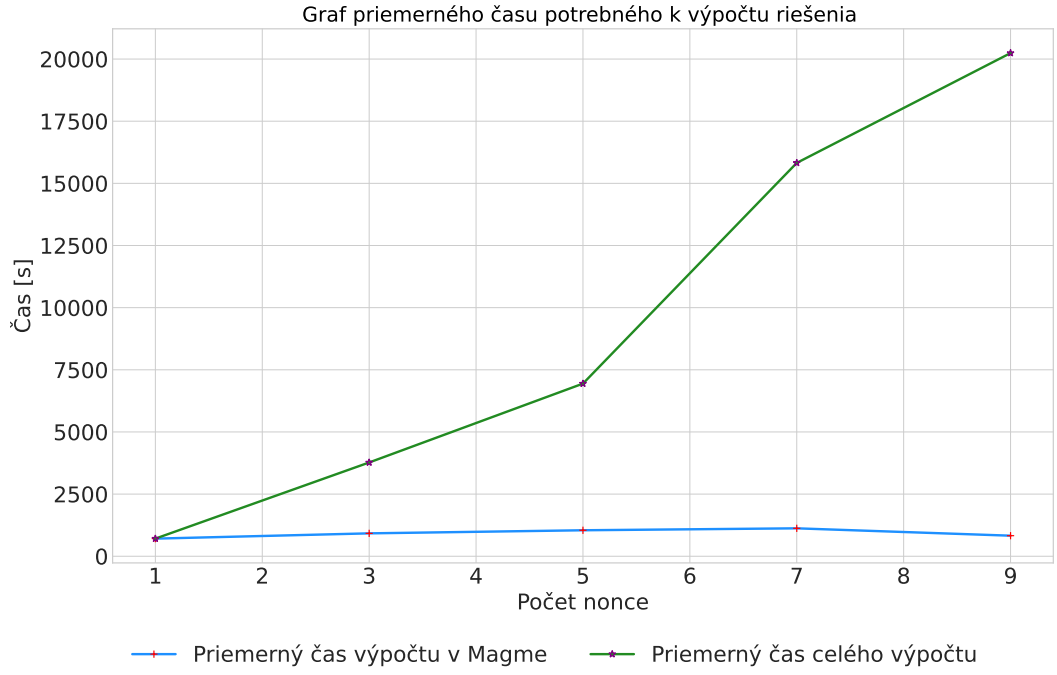
Ako ďalšiu zmenšenú variantu šifry sme vybrali 16 bitovú verziu s počtom kôl 64. Výsledky experimentov môžeme vidieť nižšie v tabuľke 5.6.

Šifra(n,r)	PR	PN	MIPM	MAPM	ČVK	CČVK
Grain(16,64)	16	1	32801	32805	708,64	709
Grain(16,64)	16	3	32412	32534	920,74	3772.,08
Grain(16,64)	16	5	32432	32494	1045	6944,5
Grain(16,64)	16	7	32408	32458	1123,3	15822,28
Grain(16,64)	16	9	32328	32434	827,3	20240,8

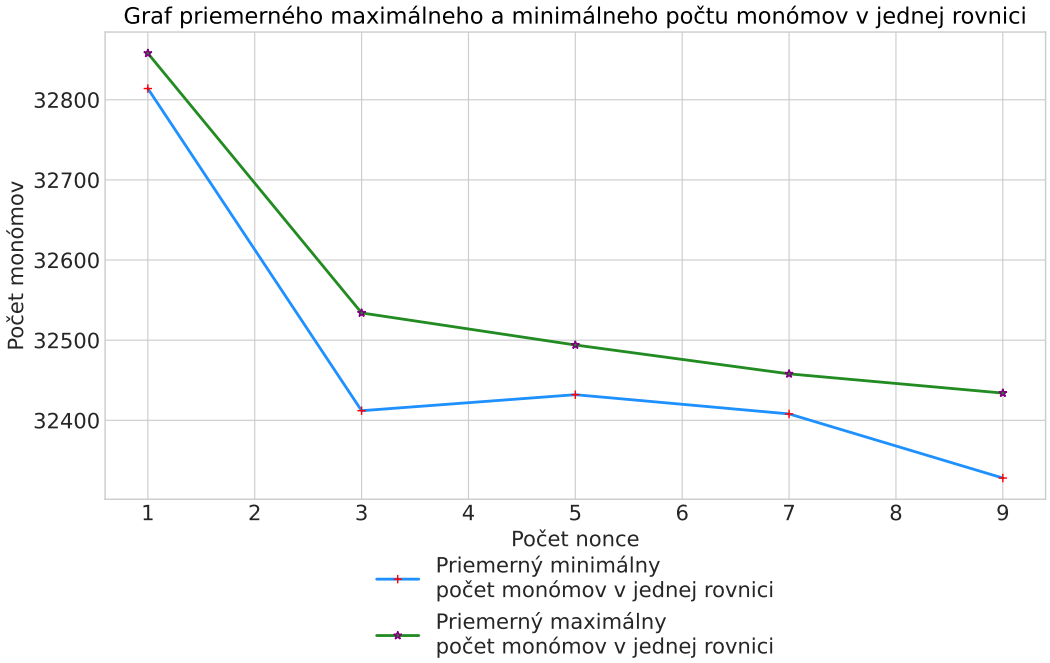
Tabuľka 5.6 Redukcia počtu polynómov pre Grain(16,64)

Z tabuľky 5.6 môžeme vidieť, že pri použití LSH a zvýšení počtu nonce sme dokázali zjednodušiť polynómy. Bohužiaľ sme ich nezjednodušili dostatočne aby sa znížil čas výpočtu Groebnerových báz a hľadania kľúča. Najlepší výsledok sme dosiahli pri použití 9 nonce, respektíve pri sústave o veľkosti 144 polynómov. Napriek tomu čas nedosahoval lepších výsledkov ako pri výpočte bez použitia LSH. Polynómy sú pri tomto počte kôl až príliš rozdielne, respektíve náhodné a LSH nie je schopné nájsť dostatočne podobné páry polynómov pri malom počte polynómov. Z toho dôvodu nedokážeme vytvoriť takú dostatočne jednoduchšiu sústavu aby sa čas výpočtu Groebnerových báz zlepšil. Môžeme vidieť len jeden beh experimentu oproti predchádzajúcim výsledkom pre Grain(16,16). Pri použití viacerých polynómov pre výpočet Groebnerových báz, než je veľkosť kľúča, nedostaneme lepšie výsledky 5.2.2.

Na grafe 5.12 môžeme vidieť čas výpočtu Groebnerových báz s hľadaním riešenia a celkový čas réžie s použitím LSH. Ďalší graf 5.13 znázorňuje vývoj maximálneho a minimálneho počtu monómov v polynóme sústavy pri zvyšujúcom sa počte nonce.



Obr. 5.12 Graf priemerného času pre Grain(16,64) pri použití LSH



Obr. 5.13 Graf priemerných počtov monómov pre Grain(16,64) pri použití LSH

5.4.3 Redukcia 24 bitovej verzie šifry Grain

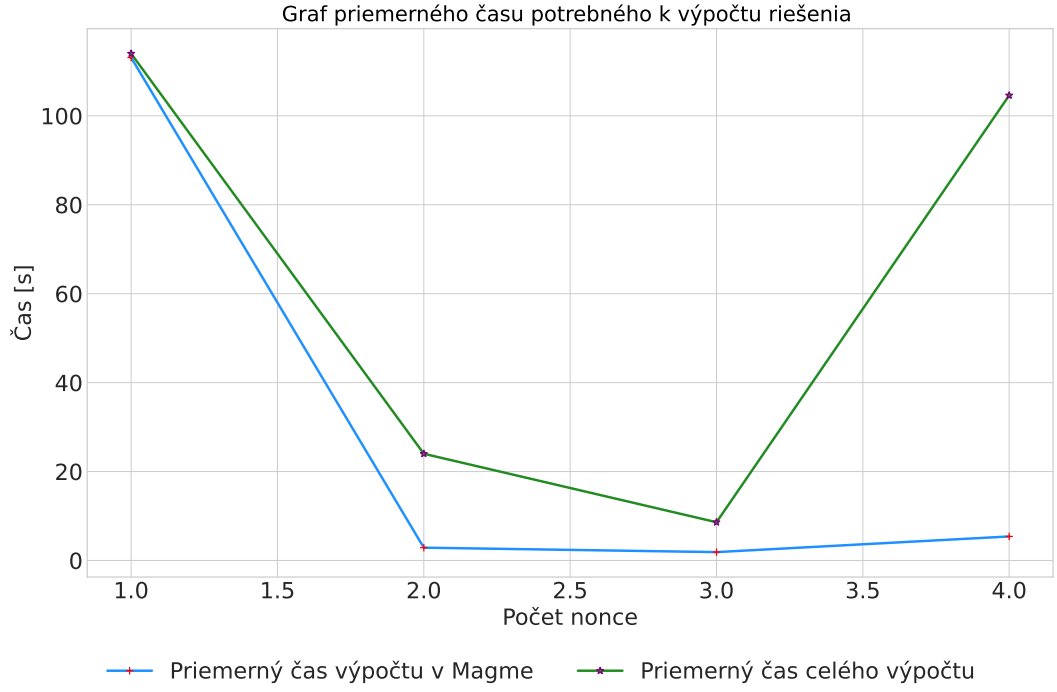
Rovnako sme vykonali experimenty pre 24 bitovú verziu šifry s počtom kôl 8. V prvom riadku tabuľky 5.7 sú výsledky pred použitím LSH. Pri tejto zmenšenej variante sme vygenerovali keystream s veľkosťou 22 a postup generovania sústavy rovníc bol rovnaký ako pri 16 bitovej verzii. Jediný rozdiel bol vtom, že sme vybrali najlepších 22 párov aby sme boli schopní nájsť riešenie výslednej Groebnerovej bázy.

Šifra(n,r)	PR	PN	MIPM	MAPM	ČVK	CČVK
Grain(24,8)	22	1	4	41869	113,1	114
Grain(24,8)	22	2	3	18	2,88	24
Grain(24,8)	22	3	3	9	1,88	8,62
Grain(24,8)	22	4	1	3	5,39	104.6

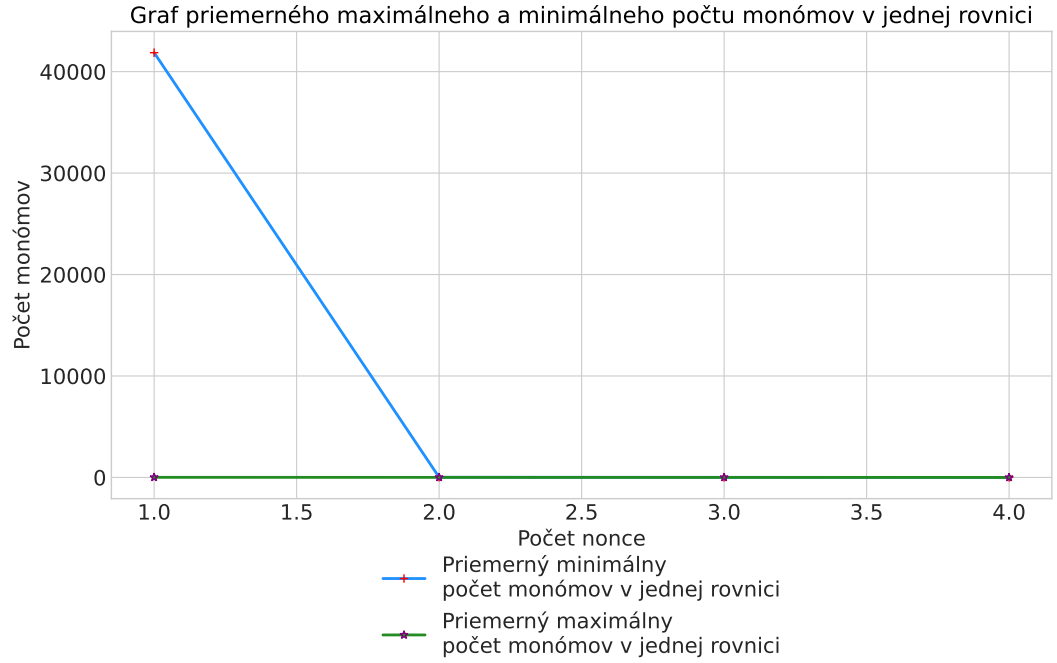
Tabuľka 5.7 Redukcia počtu polynómov pre Grain(24,8)

Po použití LSH na sústavu polynómov pre 24 bitovú variantu šifry Grain sme dosiahli zlepšenia času výpočtu kľúča a taktiež sme znížili minimálny a maximálny počet monómov v polynómoch sústavy. Na poslednom riadku tabuľky 5.7 môžeme vidieť, že sa nám čas výpočtu kľúča zhoršil. Pre túto jednoduchšiu variantu šifry mali polynómy zo sústavy príliš nízky počet monómov. Preto bol výpočet tajného kľúča neefektívny pri vyššom počte polynómov celkovej sústavy. Najefektívnejší bol pri použití 3 nonce, respektíve $3 \cdot 22$ polynómoch, z ktorých sme následne vytvorili jednoduchšiu pre riešenie Groebnerových báz.

Na grafoch 5.14 a 5.15 môžeme vidieť znázornené výsledky časov výpočtu riešenia pre danú zmenšenú variantu šifry a priemerné počty monómov.



Obr. 5.14 Graf priemerného času pre Grain(24,8) pri použití LSH



Obr. 5.15 Graf priemerných počtov monómov pre Grain(24,8) pri použití LSH

Záver

Prvým z určených pokynov bolo popísať matematický základ pre teoretickú časť našej práce. Zadefinovali sme si rôzne algebraické pojmy, ktoré viedli k definícii Groebnerových báz. Vysvetlili sme si Buchbergerov algoritmus a algoritmus F4 pre riešenie sústavy polynomiálnych rovníc. Následne sme sa zamerali na teoretický popis šifry Grain-128-AEADv2, ďalej už len Grain. Vysvetlili sme si jednotlivé stavebné bloky a funkcie šifry. Popísali sme ako funguje algoritmus pre šifrovanie a dešifrovanie.

Ďalším hlavným krokom bola praktická časť zaoberajúca sa prevodom šifry na sústavu polynomiálnych rovníc. Navrhli sme postup ako vytvoriť zmenšené varianty šifry Grain. Následne sme si vysvetlili ako daný prevod na sústavu polynomiálnych rovníc funguje. Naštudovali sme dokumentáciu jednotlivých skriptov potrebných pre generovanie sústavy rovníc a vytvorenie skriptu pre riešenie tejto sústavy pomocou Groebnerových báz.

Ďalej sme v praktickej časti vykonali rôzne experimenty, aby sme mohli vyhodnotiť rýchlosť výpočtu a potrebnú pamäť pre generovanie sústavy polynomiálnych rovníc a jej riešenia. Analyzovali sme jednotlivé výsledky pre zmenšenú 16 bitovú verziu šifry s rôznym počtom kôl. Prelomili sme 16 bitovú verziu s počtom kôl 64, kde daný počet kôl odpovedá zmenšenej 16 bitovej variante prostredníctvom nášho definovaného postupu. Pre túto variantu bol priemerný čas generovania rovníc 104,7 sekúnd a výpočtu kľúča 708,4 sekúnd. Vykonali sme experimenty aj pre niektoré ďalšie varianty šifry, pre ktoré bolo možné vygenerovať sústavu polynómov v primeranom čase. Ďalšie varianty s väčším počtom kôl sme neuviedli z časových dôvodov, nedostatočnej dĺžky keystreamu a z príliš veľkej výpočtovej zložitosti.

V rámci experimentov sme použili algoritmus LSH pre redukciu vygenerovanej sústavy polymiálnych rovníc. Stručne sme popísali jednotlivé hlavné kroky implementácie algoritmu. Sledovali sme vývoj času výpočtu tajného kľúča po zjednodušení sústavy rovníc pomocou LSH. Dané experimenty sme vykonali pre niektoré zmenšené varianty šifry. Pri 16 bitovej verzii, s počtom kôl 16, trval čas výpočtu kľúča bez použitia LSH 423 sekúnd. Najlepší dosiahnutý čas výpočtu kľúča po použití algoritmu LSH na vygenerovanú sústavy polynomiálnych rovníc trval 40,35 sekúnd. Podarilo sa nám zjednodušiť sústavy polynómiálnych rovníc a tak zrýchliť čas výpočtu tajného kľúča. Taktiež sme zistili, že pre malý počet polynómov, nie je LSH schopné nájsť dostatočne podobné polynómy pre 16 bitovú verziu šifry s počtom kôl 64. Dané polynómy sú pri vyššom počte kôl až príliš rozdielne. Algoritmus LSH sme použili aj na vygenerovanú sústavu pre 24 bitovú verziu šifry s počtom kôl 8. Čas sme znížili zo 113 sekúnd na necelé dve sekundy.

Bibliografia

1. HELL, Martin; JOHANSSON, Thomas; MAXIMOV, Alexander; MEIER, Willi; SÖNNERUP, Jonathan; YOSHIDA, Hirotaka. Grain-128AEADv2-A lightweight AEAD stream cipher. *Information Technology, Laboratory COMPUTER SECURITY RESOURCE CENTER*. 2019.
2. AGREN, Martin; HELL, Martin; JOHANSSON, Thomas; MEIER, Willi. Grain-128a: a new version of Grain-128 with optional authentication. *International Journal of Wireless and Mobile Computing*. 2011, roč. 5, č. 1, s. 48–59.
3. BARD, Gregory. *Algebraic cryptanalysis*. Springer Science & Business Media, 2009.
4. COX, David A; LITTLE, John; O'SHEA, Donal. Ideals, Varieties, and Algorithms: goAn Introduction to Computational Algebraic Geometry and Commutative Algebra. *Springer International Publishing*. 2015. Dostupné tiež z: `https://doi.org/10.1007/978-3-319-16721-3`.
5. MENEZES, Alfred J; VAN OORSCHOT, Paul C; VANSTONE, Scott A. *Handbook of applied cryptography*. CRC press, 2018.
6. GHORPADE, Sudhir R; HASAN, Sartaj Ul; KUMARI, Meena. Primitive polynomials, singer cycles and word-oriented linear feedback shift registers. *Designs, Codes and Cryptography*. 2011, roč. 58, s. 123–134.
7. SATO, Yosuke; INOUE, Shutaro; SUZUKI, Akira; NABESHIMA, Katsusuke; SAKAI, Ko. Boolean gröbner bases. *Journal of symbolic computation*. 2011, roč. 46, č. 5, s. 622–632.
8. BUCHBERGER, Bruno. Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of symbolic computation*. 2006, roč. 41, č. 3-4, s. 475–511.
9. FAUGERE, Jean-Charles. A new efficient algorithm for computing Gröbner bases (F4). *Journal of pure and applied algebra*. 1999, roč. 139, č. 1-3, s. 61–88.
10. NAIR, M Thamban; SINGH, Arindama. *Linear algebra*. Springer, 2018. Dostupné tiež z: `https://link.springer.com/book/10.1007/978-981-13-0926-7`.
11. BEIGHTON, Matthew; BARTLETT, Harry; SIMPSON, Leonie; WONG, Kenneth Koon-Ho. Algebraic Attacks on Grain-Like Keystream Generators. In: *International Conference on Information Security and Cryptology*. Springer, 2022, s. 241–270.
12. BANIK, Subhadeep; MAITRA, Subhamoy; SARKAR, Santanu. A differential fault attack on the grain family of stream ciphers. In: *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 2012, s. 122–139.
13. ZHANG, Bin; GONG, Xinxin; MEIER, Willi. Fast correlation attacks on Grain-like small state stream ciphers. *IACR Transactions on Symmetric Cryptology*. 2017, s. 58–81.

14. GHAFARI, Vahid Amin; HU, Honggang; LIN, Fujiang. On designing secure small-state stream ciphers against time-memory-data tradeoff attacks. *Cryptology ePrint Archive*. 2019.
15. GHAFARI, Vahid Amin; HU, Honggang; XIE, Chengxin. Fruit: ultralightweight stream cipher with shorter internal state. eSTREAM. *ECRYPT Stream Cipher Project*. 2016.
16. MEIER, Willi; PASALIC, Enes; CARLET, Claude. Algebraic attacks and decomposition of Boolean functions. In: *International conference on the theory and applications of cryptographic techniques*. Springer, 2004, s. 474–491.
17. PASALIC, Enes. Degree optimized resilient Boolean functions from Maiorana-McFarland class. In: *IMA International Conference on Cryptography and Coding*. Springer, 2003, s. 93–114.
18. MA, Zhen; TIAN, Tian; QI, Wen-Feng. Conditional differential attacks on Grain-128a stream cipher. *IET Information Security*. 2017, roč. 11, č. 3, s. 139–145.
19. COURTOIS, Nicolas; KLIMOV, Alexander; PATARIN, Jacques; SHAMIR, Adi. Efficient algorithms for solving overdefined systems of multivariate polynomial equations. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2000, s. 392–407.
20. BABBAGE, SH. Improved exhaustive search attacks on stream ciphers. In: *European Convention on Security and Detection, 1995.* IET, 1995, s. 161–166.
21. *Computational Algebra Group, School of Mathematics and Statistics, University of Sydney: Magma.* [online]. [cit. 2023-01-01]. Dostupné z : `http://magma.maths.usyd.edu.au/magma/documentation/`.
22. *CThe Missing Manual* [online]. [cit. 2023-04-20]. Dostupné z : `https://www.pinecone.io/learn/locality-sensitivehashing/`.
23. JANA, Berušková. *Redukováni předefinovaných systém polynomiálnich rovnic odvozených ze zjednodušených variant AES*. 2023. Dipl. pr. České vysoké učeni technické v Praze. Vypočetni a informačni centrum.
24. WANG, Jingdong; SHEN, Heng Tao; SONG, Jingkuan; JI, Jianqiu. Hashing for similarity search: A survey. *arXiv preprint arXiv:1408.2927*. 2014.

Obsah přiloženého média

- `readme.txt` — stručný popis obsahu CD
- `scripts/` — zdrojové kódy a výstupy implementácie
  - `equations/` — výstupné sústavy polynomiálnych rovníc
  - `magma_scripts/` — vstupné zdrojové kódy do Magmy
  - `magma_outputs/` — výstupné riešenia z Magmy
  - `time/` — výstupné časy pre jednotlivé skripty
  - `Grain.py` — skript obsahujúci implementáciu šifry Grain
  - `GrainPolynomial.sage` — skript pre generovanie rovníc
  - `MagmaSolver.py` — skript, ktorý vypočíta tajný kľúč
  - `Tests.py` — skript, ktorým sa spúšťajú testy
- `Graphs/` — výstupné grafy pre jednotlivé experimenty
  - `graph_plot.ipynb` — skript pre generovanie grafov
- `text/` — zdrojová forma práce vo formáte LaTeX