FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Algebraická kryptoanalýza proudových šifer založených na LFSR
Student:	Bc. Jiří Soukup
Vedoucí:	Mgr. Martin Jureček, Ph.D.
Studijní program:	Informatika
Obor / specializace:	Počítačová bezpečnost
Katedra:	Katedra informační bezpečnosti
Platnost zadání:	do konce letního semestru 2022/2023

Pokyny pro vypracování

Algebraic cryptanalysis is a modern and effective approach that consists of two steps: 1) transform a cipher into a system of polynomial equations over some finite field, 2) solve the system and find the secret key. The thesis focuses on the algebraic cryptanalysis of stream ciphers that are based on LFSR. The student will get familiar with the constructions of stream ciphers combining several LFSRs and a non-linear Boolean function and apply the Groebner basis to the polynomial equations extracted from the chosen LFSR-based stream ciphers.

Detailed instructions:
1) Describe Groebner basis and algorithms for their computation.
2) Convert at least two stream ciphers into systems of polynomial equations.
3) Apply guess-and-determine attack to the ciphers.
4) Apply a suitable program (e.g., Magma) to compute Groebner bases for the system of equations and discuss the results.

Elektronicky schválil/a prof. Ing. Róbert Lórencz, CSc. dne 25. října 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Algebraická kryptoanalýza proudových šifer založených na LFSR

Bc. Jiří Soukup

Katedra informační bezpečnosti
Vedoucí práce: Mgr. Martin Jureček, Ph.D.

2. května 2022

Poděkování

Na tomto místě bych rád poděkoval své rodině za soustavnou podporu, bez níž by tato práce nikdy nevznikla, pánům MČ a SP za pravidelnou psychickou vzpruhu a v neposlední řadě Mgr. Martinu Jurečkovi, Ph.D. za cenné rady a vstřícnost, s níž je poskytoval.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principu při přípravě vysokoškolských závěrečných prací.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona, ve znění pozdějších předpisu, zejména skutečnost, že České vysoké učení technické v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 citovaného zákona.

V Praze dne 2. května 2022 . . . . . .

České vysoké učení technické v Praze
Fakulta informačních technologií

Odkaz na tuto práci

Soukup, Jiří. *Algebraická kryptoanalýza proudových šifer založených na LFSR.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022.

Abstrakt

Keystream produkovaný proudovou šifrou založenou na LFSR je možné popsat soustavou polynomiálních rovnic. Nalezením řešení takové soustavy je šifra prolomena. Za pomoci algoritmu F4 lze soustavu polynomiálních rovnic efektivně vyřešit. Při znalosti parametrů proudové šifry založené na LFSR může být výpočet dále urychlen použitím metody guess-and-determine.

V této práci rozebereme, jak algoritmus F4 funguje, a pokusíme se najít vhodné způsoby využití metody guess-and-determine pro jeho urychlení při řešení soustavy polynomiálních rovnic odpovídající proudové šifře založené na LFSR.

Klíčová slova Gröbnerovy báze, algoritmus F4, proudová šifra, LFSR, guess-and-determine

Abstract

Keystream produced by LFSR-based stream ciphers can be described by a system of polynomial equations. Such ciphers can be cracked by solving the system. Using the F4 algorithm we can effectively solve a system of polynomial equations. Knowing the parameters of an LFSR-based stream cipher we can further speed up the computation by applying the guess-and-determine method.

In this thesis, we describe how the F4 algorithm works. And mainly we try to find effective ways of applying the guess-and-determine method to reduce the computation time of the F4 algorithm when used to solve a system of polynomial equations that represent an LFSR-based stream cipher.

Keywords Gröbner bases, F4 algorithm, stream cipher, LFSR, guess-and-determine

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Jeden ze způsobů, jakými jsou konstruovány proudové šifry, využívá jako dílčí stavební prvek LFSR. Keystream produkovaný proudovými šiframi tohoto typu lze přirozeně popsat soustavou polynomiálních rovnic. Pokud v rámci útoku typu *known-plaintext* vyřešíme tuto soustavu, získáme informaci, z níž můžeme zrekonstruovat tajný klíč, který byl použit při generování keystreamu.

Gröbnerovy báze poskytují efektivní způsob řešení soustav polynomiálních rovnic. S jejich pomocí lze najít ekvivalentní soustavu, jejíž řešení je výrazně jednodušší než řešení soustavy původní. Algoritmus F4 je jedním z algoritmů, které jsou schopny Gröbnerovu bázi odpovídající soustavě polynomiálních rovnic efektivně najít. Je tedy vhodným kandidátem pro použití při tomto útoku.

Řešení soustavy polynomiálních rovnic je možné zrychlit metodou *guess-and-determine*. Tato metoda „hádáním“ některých neznámých v soustavě rozkládá původní problém na několik dílčích podproblémů. Při vhodné volbě odhadovaných proměnných můžeme dosáhnout výrazně kratšího času řešení původní soustavy polynomiálních rovnic, a tedy i času celého útoku.

Hlavním cílem této práce bude nalezení efektivního způsobu využití metody *guess-and-determine* při řešení soustavy polynomiálních rovnic popisující keystream proudové šifry založené na LFSR.

V kapitole 1 vystavíme teorii potřebnou pro zavedení Gröbnerových bází. Popíšeme a vysvětlíme algoritmus F4. V závěru kapitoly ukážeme, jak lze pomocí Gröbnerových bází řešit soustavu polynomiálních rovnic. Kapitola 2 shrnuje, co je LFSR a jak s jeho pomocí můžeme zkonstruovat proudovou šifru. Také obsahuje vysvětlení převodu proudové šifry založené na LFSR na soustavu polynomiálních rovnic. Kapitola 3 popisuje provedená měření a jejich výsledky. Pro tři různé proudové šifry zde testujeme několik variant využití metody *guess-and-determine* pro řešení příslušné soustavy polynomiálních rovnic. V kapitole 4 zdokumentujeme software, který vznikl pro měření popsaná v kapitole 3, a vysvětlíme, jak funguje. A na závěr v kapitole 5 tuto práci zasadíme do kontextu jiných prací, které se jí tematicky dotýkají.

Kapitola 1

Gröbnerovy báze

Jedním ze základních prostředků pro výpočet soustav polynomiálních rovnic, kterými se dále v této práci budeme zabývat, jsou *Gröbnerovy báze*. Jejich zavedení a popisu bude věnována tato kapitola.

Nebude-li uvedeno jinak, všechny definice a tvrzení v této kapitole budou vycházet z [2].

1.1 Polynomy v n proměnných

Definice 1.1.1. Pojmem *monom* označíme součin ve tvaru

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

kde exponenty $\alpha_1, \ldots, \alpha_n$ jsou nezáporná celá čísla. Za *celkový stupeň* monomu pak budeme považovat součet $\alpha_1 + \cdots + \alpha_n$.

Poznámka 1.1.2. Pro zápis monomu budeme používat zjednodušenou notaci. Ať $\alpha = (\alpha_1, \ldots, \alpha_n)$ je n-tice nezáporných celých čísel. Pak položíme

$$x^\alpha = x_1^{\alpha_1} \cdot x_2^{\alpha_2} \cdots x_n^{\alpha_n}.$$

Pro $\alpha = (0, \ldots, 0)$ platí $x^\alpha = 1$. Zároveň $|\alpha| = \alpha_1 + \cdots + \alpha_n$ bude značit celkový stupeň monomu.

Definice 1.1.3. *Polynom* f v proměnných $x_1, \ldots, x_n$ nad tělesem k je konečná lineární kombinace monomů s koeficienty v k. Polynom f zapíšeme jako

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

V sumě výše sčítáme přes konečný počet n-tic $\alpha = (\alpha_1, \ldots, \alpha_n)$. Množinu všech polynomů v proměnných $x_1, \ldots, x_n$ nad tělesem k označíme $k[x_1, \ldots, x_n]$.

Poznámka 1.1.4. Množina $k[x_1, \ldots, x_n]$ z definice 1.1.3 je *okruh*.

Definice 1.1.5. Ať $f = \sum_{\alpha} a_\alpha x^\alpha$ je polynom z $k[x_1, \ldots, x_n]$.

(i) a_α nazveme *koeficient* monomu x^α.

(ii) Pokud $a_\alpha \neq 0$, pak $a_\alpha x^\alpha$ označíme jako *term* polynomu f.

(iii) *Celkový stupeň* nenulového polynomu f je největší $|\alpha|$ takové, že koeficient a_α je nenulový. Celkový stupeň polynomu f budeme značit $\deg(f)$. Celkový stupeň nulového polynomu nedefinujeme.

1.2 Monomiální uspořádání

V případě polynomů v jedné proměnné existuje přirozený způsob, jak jednotlivé monomy polynomu řadit a jak díky tomu pracovat s pojmy jako je *stupeň* nebo třeba *vedoucí koeficient* polynomu. Abychom měli stejné možnosti u polynomů ve vícero proměnných, s nimiž v této práci budeme hojně pracovat, je zapotřebí rozvést teorii *monomiálních uspořádání*.

Definice 1.2.1. Relace $>$ na $\mathbb{N}_0^n$ je *lineární uspořádání*, pokud pro každou dvojici $\alpha, \beta \in \mathbb{N}_0^n$ platí právě jedno ze tří tvrzení

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

Definice 1.2.2. Relace $>$ na $\mathbb{N}_0^n$ je *dobré uspořádání*, pokud každá neprázdná podmnožina $\mathbb{N}_0^n$ má nejmenší prvek vzhledem k $>$. Tj. pro každou neprázdnou množinu $A \subseteq \mathbb{N}_0^n$ existuje $\alpha \in A$ takové, že $\beta > \alpha$ pro každé $\beta \in A, \beta \neq \alpha$.

Definice 1.2.3. *Monomiální uspořádání* $>$ na $k[x_1, \ldots, x_n]$ je relace na $\mathbb{N}_0^n$, která splňuje

(i) $>$ je lineární uspořádání na $\mathbb{N}_0^n$,

(ii) pokud $\alpha > \beta$ a $\gamma \in \mathbb{N}_0^n$, pak $\alpha + \gamma > \beta + \gamma$,

(iii) $>$ je dobré uspořádání na $\mathbb{N}_0^n$.

Poznámka 1.2.4. K definicím 1.2.1, 1.2.2 a 1.2.3 navíc dodejme, že

- množinou $\mathbb{N}_0^n$ myslíme množinu všech n-tic $(\alpha_1, \ldots, \alpha_n)$ takových, že α_i je nezáporné celé číslo pro všechna $i \in \{1, \ldots, n\}$,
- popsaná uspořádání mohou být ekvivalentně zavedena jako relace na množině monomů $x^\alpha, \alpha \in \mathbb{N}_0^n$, splňující analogické požadavky.

Dále uvedeme několik příkladů monomiálních uspořádání.

Definice 1.2.5. Ať $\alpha = (\alpha_1, \ldots, \alpha_n)$ a $\beta = (\beta_1, \ldots, \beta_n)$ jsou z $\mathbb{N}_0^n$. Řekneme, že $\alpha >_{lex} \beta$, pokud zleva první nenulový prvek rozdílu $\alpha - \beta \in \mathbb{Z}^n$ je kladný. Budeme psát $x^\alpha >_{lex} x^\beta$, pokud $\alpha >_{lex} \beta$. Relaci $>_{lex}$ nazveme *lexikografické uspořádání.*

Příklad 1.2.6.

- $\alpha = (2, 0, 4) >_{lex} \beta = (1, 5, 5)$, jelikož $\alpha - \beta = (1, -5, -1)$
- $\alpha = (2, 0, 4) >_{lex} \beta = (2, 0, 3)$, jelikož $\alpha - \beta = (0, 0, 1)$

Definice 1.2.7. Ať $\alpha, \beta \in \mathbb{N}_0^n$. Řekneme, že $\alpha >_{grlex} \beta$, pokud

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

nebo

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

Relaci $>_{grlex}$ nazveme *odstupňované lexikografické uspořádání.*

Poznámka 1.2.8. U odstupňovaného lexikografického uspořádání tedy získává prioritu celkový stupeň monomu.

Příklad 1.2.9.

- $\alpha = (0, 4, 4) >_{grlex} \beta = (1, 2, 2)$, jelikož $|\alpha| > |\beta|$
- $\alpha = (1, 3, 4) >_{grlex} \beta = (0, 4, 4)$, jelikož $|\alpha| = |\beta|$ a $\alpha - \beta = (1, -1, 0)$, tj. $\alpha >_{lex} \beta$

Definice 1.2.10. Ať $\alpha, \beta \in \mathbb{N}_0^n$. Řekneme, že $\alpha >_{grevlex} \beta$, pokud

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

nebo

$$|\alpha| = |\beta| \quad \text{a zprava první nenulový prvek rozdílu } \alpha - \beta \in \mathbb{Z}^n \text{ je záporný.}$$

Relaci $>_{grevlex}$ nazveme *obrácené odstupňované lexikogracifké uspořádání.*

Příklad 1.2.11.

- $\alpha = (0, 4, 4) >_{grevlex} \beta = (1, 2, 2)$, jelikož $|\alpha| > |\beta|$
- $\alpha = (2, 2, 2) >_{grevlex} \beta = (3, 0, 3)$, jelikož $|\alpha| = |\beta|$ a $\alpha - \beta = (-1, 2, -1)$, tj. zprava první nenulový prvek je -1.

  Zároveň si ale všimněme, že $\beta >_{grlex} \alpha$, jelikož $|\beta| = |\alpha|$ a $\beta >_{lex} \alpha$.

Lemma 1.2.12. *Relace $>_{lex}$, $>_{grlex}$ i $>_{grevlex}$ jsou monomiální uspořádání na $\mathbb{N}_0^n$.*

Definice 1.2.13. Ať $f = \sum_{\alpha} a_\alpha x^\alpha$ je nenulový polynom z $k[x_1, \ldots, x_n]$ a ať $>$ je monomiální uspořádání.

(i) *Multistupeň* polynomu f je

$$\text{multideg}(f) = \max\left(\{\alpha \in \mathbb{N}_0^n \mid a_\alpha \neq 0\}\right),$$

kde maximum je uvažováno vzhledem k uspořádání $>$.

(ii) *Vedoucí koeficient* polynomu f je

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

(iii) *Vedoucí monom* polynomu f je

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

(iv) *Vedoucí term* polynomu f je

$$\text{LT}(f) = \text{LC}(f) \cdot \text{LM}(f).$$

Příklad 1.2.14. Mějme polynom $f = 2xy^3z^4 - 3y^4z^4 + xy^2z^2 + 1$, uvažujme odstupňované lexikografické uspořádání $>_{grlex}$ a pro uspořádání proměnných x, y, z ať platí $x > y > z$. Vzhledem k $>_{grlex}$ pak máme

- $\text{multideg}(f) = (1, 3, 4)$,
- $\text{LC}(f) = 2$,
- $\text{LM}(f) = xy^3z^4$,
- $\text{LT}(f) = 2xy^3z^4$.

1.3 Dělení polynomů v n proměnných

Se znalostí monomiálních uspořádání můžeme nad polynomy v n proměnných zavést dělení polynomu množinou polynomů. Algoritmus dělení zde uvádět nebudeme, případě zájmu je k nalezení v [2] jako součást důkazu věty o dělení v $k[x_1, \ldots, x_n]$ (strana 65). Větu samotnou zde pro její důležitost uvedeme.

Věta 1.3.1 (Dělení v $k[x_1, \ldots, x_n]$)**.** *Ať $>$ je monomiální uspořádání na $\mathbb{N}_0^n$ a ať $F = (f_1, \ldots, f_s)$ je uspořádaná s-tice polynomů z $k[x_1, \ldots, x_n]$. Pak každé $f \in k[x_1, \ldots, x_n]$ můžeme zapsat ve tvaru*

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

kde $\forall i \in \{1, \ldots, s\} : q_i \in k[x_1, \ldots, x_n]$ a $r \in k[x_1, \ldots, x_n]$. Zároveň buďto $r = 0$, nebo r je lineární kombinací (s koeficienty v k) monomů, z nichž žádný není dělitelný žádným $\mathrm{LT}(f_i)$ pro $i \in \{1, \ldots, s\}$. Pro $i \in \{1, \ldots, s\}$ navíc $q_i f_i \neq 0$ implikuje

$$\mathrm{multideg}(f) > \mathrm{multideg}(q_i f_i) \quad \textit{nebo} \quad \mathrm{multideg}(f) = \mathrm{multideg}(q_i f_i).$$

Definice 1.3.2. Hodnotu r z věty 1.3.1 výše nazveme *zbytkem po dělení polynomu f uspořádanou s-ticí F* a budeme ji značit $\overline{f}^F$.

1.4 Polynomiální ideály

Definice 1.4.1. Mějme podmnožinu $I \subseteq k[x_1, \ldots, x_n]$. I je *ideál* $k[x_1, \ldots, x_n]$, pokud

(i) $0 \in I$,

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

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

Poznámka 1.4.2.

- V bodu (i) definice 1.4.1 výše 0 značí nulový polynom z $k[x_1, \ldots, x_n]$.
- Ideál lze definovat obecněji pro libovolný *okruh*, nejen pro okruh polynomů. V kontextu této práce nám ovšem postačí definice výše.
- Pojmem *polynomiální ideál* rozumíme ideál nad okruhem polynomů.

Definice 1.4.3. Ať $f_1, \ldots, f_s$ jsou polynomy z $k[x_1, \ldots, x_n]$, pak položíme

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

Lemma 1.4.4. *Mějme $f_1, \ldots, f_s \in k[x_1, \ldots, x_n]$, pak $\langle f_1, \ldots, f_s \rangle$ je ideálem $k[x_1, \ldots, x_n]$. Tento ideál nazveme* ideál generovaný *$f_1, \ldots, f_s$.*

Definice 1.4.5. Řekneme, že ideál $I \subseteq k[x_1, \ldots, x_n]$ je *monomiální ideál*, pokud existuje množina $A \subseteq \mathbb{N}_0^n$ taková, že I sestává ze všech polynomů, které jsou konečnou sumou tvaru

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

Pak budeme psát $I = \langle x^\alpha \mid \alpha \in A \rangle$.

Lemma 1.4.6. *Ať $I = \langle x^\alpha \mid \alpha \in A\rangle$ je monomiální ideál. Pak monom x^β je prvkem I právě tehdy, když $x^\alpha \mid x^\beta$ pro nějaké $\alpha \in A$.*

Příklad 1.4.7. Mějme monomiální ideál $I = \langle x^3y, xy^2\rangle$. Potom

- $x^3y^2 \in I$, jelikož $x^3y^2 = x^3y \cdot y$, tj. $x^3y \mid x^3y^2$,
- $x^2y \notin I$, jelikož $x^3y \not| \, x^2y$ a $xy^2 \not| \, x^2y$.

Definice 1.4.8. Ať $I \subseteq k[x_1, \ldots, x_n]$ je ideál různý od $\{0\}$. Zafixujme monomiální uspořádání na $k[x_1, \ldots, x_n]$. Pak

(i) výrazem $\mathrm{LT}(I)$ označíme množinu vedoucích termů nenulových prvků v I. Tedy

$$\mathrm{LT}(I) = \{cx^\alpha \mid \exists f \in I \setminus \{0\} : \mathrm{LT}(f) = cx^\alpha\},$$

(ii) výrazem $\langle \mathrm{LT}(I)\rangle$ označíme ideál generovaný prvky množiny $\mathrm{LT}(I)$.

Věta 1.4.9 (Hilbertova věta o bázi)**.** *Každý ideál $I \subseteq k[x_1, \ldots, x_n]$ má konečnou generující množinu. Tedy $I = \langle g_1, \ldots, g_t\rangle$ pro nějaké $g_1, \ldots, g_t \in I$.*

1.4.1 Ideály a variety

Definice 1.4.10. Ať $f_1, \ldots, f_s$ jsou polynomy z $k[x_1, \ldots, x_n]$. Pak položíme

$$\mathrm{V}(f_1, \ldots, f_s) = \{(a_1, \ldots, a_n) \in k^n \mid \forall i \in \{1, \ldots, s\} : f_i(a_1, \ldots, a_n) = 0\}.$$

$\mathrm{V}(f_1, \ldots, f_s)$ nazveme *afinní varietou* danou $f_1, \ldots, f_s$.

Poznámka 1.4.11. Všimněme si, že varieta je právě množina řešení soustavy polynomiálních rovnic dané polynomy $f_1, \ldots, f_s$.

Definice 1.4.12. Ať $V \subseteq k^n$ je afinní varieta. Potom položíme

$$\mathrm{I}(V) = \{f \in k[x_1, \ldots, x_n] \mid \forall (a_1, \ldots, a_n) \in V : f(a_1, \ldots, a_n) = 0\}.$$

Tvrzení 1.4.13. *Pokud $V \subseteq k^n$ je afinní varieta, pak*
$\mathrm{I}(V) \subseteq k[x_1, \ldots, x_n]$ *je ideál. Ideál* $\mathrm{I}(V)$ *nazveme* ideál variety V.

Definice 1.4.14. Mějme ideál $I \subseteq k[x_1, \ldots, x_n]$. Potom položíme

$$\mathrm{V}(I) = \{(a_1, \ldots, a_n) \in k^n \mid \forall f \in I : f(a_1, \ldots, a_n) = 0\}.$$

Tvrzení 1.4.15. $\mathrm{V}(I)$ *je afinní varieta. Speciálně pokud $I = \langle f_1, \ldots, f_s\rangle$, pak*

$$\mathrm{V}(I) = \mathrm{V}(f_1, \ldots, f_s).$$

Poznámka 1.4.16. Důležitým důsledkem tvrzení 1.4.15 je to, že variety jsou určeny ideály. Máme-li soustavu polynomiálních rovnic, množina polynomů této soustavy generuje polynomiální ideál. Vezmeme-li jinou množinu polynomů takovou, že generuje stejný ideál, a vyřešíme-li odpovídající soustavu polynomiálních rovnic, získáme řešení, které je totožné s řešením soustavy původní.

1.5 Gröbnerova báze

Definice 1.5.1. Zafixujme monomiální uspořádání na okruhu polynomů $k[x_1, \ldots, x_n]$. Konečnou podmnožinu $G = \{g_1, \ldots, g_t\}$ ideálu $I \subseteq k[x_1, \ldots, x_n]$ různého od $\{0\}$ nazveme *Gröbnerova báze* I, pokud

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

Dále položíme $\langle \emptyset \rangle = \{0\}$, čímž prázdnou množinu $\emptyset$ učiníme Gröbnerovou bází triviálního ideálu $\{0\}$.

Příklad 1.5.2. Uvažujme monomiální uspořádání $>_{lex}$, kde $x > y$ a ideál

$$I = \langle xy + 2x, x^2 + y^2 \rangle$$

nad okruhem $\mathbb{F}_5[x, y]$. Potom množina $G = \{xy + 2x, x^2 + y^2, y^3 + 2y^2\}$ je Gröbnerovou bází ideálu I.

Věta 1.5.3. *Zafixujme monomiální uspořádání na $k[x_1, \ldots, x_n]$. Pak každý ideál $I \subseteq k[x_1, \ldots, x_n]$ má Gröbnerovu bázi. Navíc jakákoli Gröbnerova báze ideálu I je bází I.*

Poznámka 1.5.4. V tomto kontextu bází ideálu I rozumíme množinu polynomů, která I generuje.

Tvrzení 1.5.5. *Ať $I \subseteq k[x_1, \ldots, x_n]$ je ideál a $G = \{g_1, \ldots, g_t\}$ je Gröbnerova báze I. Pak pro dané $f \in [k_1, \ldots, x_n]$ existuje jednoznačně určené $r \in k[x_1, \ldots, x_n]$ takové, že*

(i) žádný term r není dělitelný žádným z $\mathrm{LT}(g_1), \ldots, \mathrm{LT}(g_t)$,

(ii) existuje $g \in I$ takové, že $f = g + r$.

Speciálně r je zbytkem po dělení polynomu f t-ticí polynomů z G bez ohledu na to, jak jsou v G uspořádané.

Poznámka 1.5.6. Tvrzení 1.5.5 výše se mezi doposud uvedenými informacemi dotýká zejména věty 1.3.1 a definice 1.3.2. U nich nám říká, že dělíme-li polynom s-ticí, která je Gröbnerovou bází, můžeme tuto s-tici uvažovat jako množinu, tj. neuspořádanou.

Definice 1.5.7. Mějme polynomiální ideál I a jeho Gröbnerovu bázi G. Řekneme, že G je *redukovaná Gröbnerova báze* I, pokud

(i) $\forall p \in G : \mathrm{LC}(p) = 1$,

(ii) $\forall p \in G$: žádný monom z p neleží v $\langle \mathrm{LT}(G \setminus \{p\}) \rangle$.

Věta 1.5.8. *Mějme zafixované monomiální uspořádání. $I \neq \{0\}$ ať je polynomiální ideál. Potom I má právě jednu redukovanou Gröbnerovu bázi.*

Poznámka 1.5.9. Máme-li Gröbnerovu bázi G ideálu I, můžeme z ní získat *redukovanou* Gröbnerovu bázi ideálu I postupnou redukcí všech prvků $g \in G$ modulo $G \setminus \{g\}$ (viz větu 1.3.1) [3].

Co je Gröbnerova báze, již je definováno. V tom okamžiku se nabízí otázka, jak najít Gröbnerovu bázi daného ideálu. Z věty 1.4.9 víme, že každý ideál lze popsat nějakou jeho konečnou bází. Jedná se tedy o problém nalezení Gröbnerovy báze odpovídající libovolné (konečné) bázi ideálu. Zdůrazněme, že v kontextu řešení soustavy polynomiálních rovnic může být jak výchozí báze ideálu, tak nalezená Gröbnerova báze vskutku libovolná (viz poznámku 1.4.16 a větu 1.5.3).

Pro řešení tohoto problému vzniklo vícero algoritmů (viz [2, kapitoly 2 a 10]). Podrobně se v této práci zaměříme jen na jeden z nich, a to na algoritmus *F4*, který je použit v praktické části práce.

Dílčím podproblémem, se kterým se algoritmy pro hledání Gröbnerových bází typicky setkávají, je potřeba rozeznat, zda daná množina polynomů je Gröbnerovou bází daného ideálu. Tomuto podproblému se budeme věnovat nejprve.

Definice 1.5.10. Ať $f, g \in k[x_1, \ldots, x_n]$ jsou nenulové polynomy.

(i) Pokud $\text{multideg}(f) = \alpha$ a $\text{multideg}(g) = \beta$, pak položme $\gamma = (\gamma_1, \ldots, \gamma_n)$, kde
$$\gamma_i = \max(\{\alpha_i, \beta_i\}) \quad \forall i \in \{1, \ldots, n\}.$$
Monom x^γ nazveme *nejmenším společným násobkem* $\text{LM}(f)$ a $\text{LM}(g)$ a budeme psát $x^\gamma = \text{lcm}(\text{LM}(f), \text{LM}(g))$.

(ii) Jako *S-polynom* f a g označíme polynom
$$\text{S}(f, g) = \frac{x^\gamma}{\text{LT}(f)} \cdot f - \frac{x^\gamma}{\text{LT}(g)} \cdot g.$$

Poznámka 1.5.11. Podíváme-li se zpět na definici 1.5.1, vidíme, že aby báze $G = \{g_1, \ldots, g_t\}$ ideálu $I \subseteq k[x_1, \ldots, x_n]$ *nebyla* jeho Gröbnerovou bází, musí existovat $f \in I$ takové, že
$$\text{LT}(f) \notin \langle \text{LT}(g_1), \ldots, \text{LT}(g_t) \rangle. \tag{1.1}$$

Toto nastane, pokud pro $g_i, g_j \in G$ ideál I obsahuje vhodnou kombinaci tvaru
$$h = ax^\alpha g_i - bx^\beta g_j \tag{1.2}$$
takovou, že vedoucí termy obou polynomů v rozdílu v (1.2) se odečtou a zůstanou jen termy menší (vzhledem k uvažovanému monomiálnímu uspořádání). Zároveň polynom h z (1.2) je opravdu prvkem I, jelikož g_i i g_j jsou prvky báze I.

Všimněme si, že S-polynom zavedený v definici 1.5.10 je zkonstruován právě tak, aby došlo k výše popsanému odečtení vedoucích termů a abychom tedy potenciálně mohli získat prvek $f \in I$ splňující (1.1).

Věta 1.5.12 (Buchbergerovo kritérium)**.** *Mějme polynomiální ideál I a zafixujme nějaké monomiální uspořádání. Pak báze $G = \{g_1, \ldots, g_t\}$ ideálu I je jeho Gröbnerovou bází právě tehdy, když pro každou dvojici $i \neq j$ je*

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

Poznámka 1.5.13.

- Věta 1.5.12 jinými slovy říká, že zbytek po dělení polynomu $\mathrm{S}(g_i, g_j)$ množinou G musí být nulový (viz definici 1.3.2) pro každou dvojici $i \neq j$.
- Věta 1.5.12 je základem *Buchbergerova algoritmu* (viz [2, kapitola 2.7]) pro výpočet Gröbnerových bází. Jak již bylo výše poznamenáno, tímto algoritmem se blíže zabývat nebudeme.

Definice 1.5.14. Zafixujme nějaké monomiální uspořádání, mějme $f \in k[x_1, \ldots, x_n]$ a ať $G = \{g_1, \ldots, g_t\} \subseteq k[x_1, \ldots, x_n]$. Řekneme, že f se *redukuje na nulu modulo G*, psáno

$$f \to_G 0,$$

pokud f má *standardní reprezentaci* tvaru

$$f = A_1 g_1 + \cdots + A_t g_t,$$

kde

$$A_i \in k[x_1, \ldots, x_n] \quad \forall i \in \{1, \ldots, t\}.$$

Věta 1.5.15. *Mějme polynomiální ideál I a zafixujme nějaké monomiální uspořádání. Báze $G = \{g_1, \ldots, g_t\}$ ideálu I je jeho Gröbnerovou bází právě tehdy, když pro každou dvojici $i \neq j$ platí*

$$\mathrm{S}(g_i, g_j) \to_G 0.$$

Poznámka 1.5.16. Všimněme si, že pokud pro $f \in k[x_1, \ldots, x_n]$ a t-tici $G = (g_1, \ldots, g_t)$ platí

$$\overline{f}^G = 0,$$

pak z definice 1.3.2 a věty 1.3.1 je

$$f = q_1 g_1 + \cdots + q_t g_t + 0$$

pro nějaká $q_1, \ldots, q_t \in k[x_1, \ldots, x_n]$, a tedy

$$f \to_G 0.$$

Opačná implikace obecně neplatí. Jako protipříklad uvažme lexikografické uspořádání, kde $x > y$, a polynomy

$$\begin{aligned} f &= 2x^2y + 2xy + y, \\ g_1 &= xy + 2y, \\ g_2 &= x^2 + 1, \end{aligned}$$

nad $\mathbb{F}_3[x, y]$. Položíme-li $G = (g_1, g_2)$, pak

$$f \to_G 0,$$

jelikož

$$f = x \cdot g_1 + y \cdot g_2.$$

Současně platí

$$f = (2x + 1) \cdot g_1 + 0 \cdot g_2 + 2y,$$

přičemž

$$\mathrm{LT}(g_1) = xy \not| \, y \quad \text{a} \quad \mathrm{LT}(g_2) = x^2 \not| \, y.$$

Tudíž ani $\mathrm{LT}(g_1)$, ani $\mathrm{LT}(g_2)$ nedělí žádný z monomů obsažených v polynomu $r = 2y$. Proto můžeme psát

$$\overline{f}^G = 2y \neq 0$$

(viz větu 1.3.1). Nalezli jsme tedy f a G takové, že

$$f \to_G 0 \quad \text{a zároveň} \quad \overline{f}^G \neq 0,$$

čímž je protipříklad dokončen.

Poznámka 1.5.17. Věty 1.5.15 je využito v algoritmu F4, jehož popis bude obsahem následující sekce.

1.6 Algoritmus F4

Kdykoli budeme hovořit o F4, je vhodné mít na paměti, že toto označení popisuje celou skupinu algoritmů, které se od sebe ve větší či menší míře liší. Všechny ale mají společnou myšlenku. A tu se v této sekci budeme snažit zachytit.

Algoritmus 1.6.1 popisuje F4 pomocí pseudokódu. Nyní se podívejme podrobněji na význam jednotlivých jeho částí.

1.6.1 s-tice F

F je vstupní s-tice polynomů, které tvoří bázi ideálu, jehož Gröbnerovu bázi hledáme.

Algoritmus 1.6.1 F4

Vstup: $F = (f_1, \ldots, f_s)$
Výstup: G takové, že G je Gröbnerovou bází ideálu $I = \langle f_1, \ldots, f_s \rangle$

1: $G \leftarrow F$
2: $t \leftarrow s$
3: $B \leftarrow \{\{i, j\} \mid 1 \leq i < j \leq s\}$
4: **while** $B \neq \emptyset$ **do**
5: zvol $B' \neq \emptyset$, $B' \subseteq B$
6: $B \leftarrow B \setminus B'$
7: $L \leftarrow \bigcup_{\{i,j\} \in B'} \left\{ \frac{\mathrm{lcm}(\mathrm{LM}(f_i), \mathrm{LM}(f_j))}{\mathrm{LT}(f_i)} \cdot f_i, \frac{\mathrm{lcm}(\mathrm{LM}(f_i), \mathrm{LM}(f_j))}{\mathrm{LT}(f_j)} \cdot f_j \right\}$
8: $M \leftarrow \textsc{SpočtiM}(L, G)$
9: $N \leftarrow$ redukovaný odstupňovaný tvar matice M
10: $N^+ \leftarrow \{n \in \pi(N) \mid \mathrm{LM}(n) \notin \langle \mathrm{LM}(\pi(M)) \rangle\}$
11: **for** $n \in N^+$ **do**
12: $t \leftarrow t + 1$
13: $f_t \leftarrow n$
14: $G \leftarrow G \cup \{f_t\}$
15: $B \leftarrow B \cup \{\{i, t\} \mid 1 \leq i < t\}$
16: **return** G

1.6.2 Množina G, proměnná t

Množina G je výstupem algoritmu, tj. je to hledaná Gröbnerova báze. Zároveň G slouží jako pracovní množina. Na začátku jsou v G obsaženy právě ty polynomy, které jsou v F. Postupně jsou do G přidávány polynomy nové.

Proměnná t označuje aktuální počet prvků množiny G.

1.6.3 Množiny B a B'

B je množina neuspořádaných dvojic indexů $\{k_1, k_2\}$ odpovídajících jednotlivým prvkům f_{k_1}, f_{k_2} množiny G. V B jsou obsaženy ty dvojice indexů $\{i, j\}$, pro které *není* ověřeno, že

$$\mathrm{S}(f_i, f_j) \to_G 0. \tag{1.3}$$

Toto má přímou souvislost s větou 1.5.15! Celý běh algoritmu 1.6.1 tak lze dále zjednodušit do následujících kroků.

1. Pokud $B = \emptyset$, tj. pokud jsme ověřili (1.3) pro všechny dvojice $\{f_i, f_j\} \in G$, vrať G.
2. Zvol $B' \subseteq B$ a její prvky z B odeber.

3. Doplň do G polynomy tak, aby (1.3) platilo pro všechny dvojice $\{f_i, f_j\} \in G$ odpovídající prvkům $\{i, j\} \in B'$.

4. Doplň do B nové dvojice indexů. Tyto odpovídají prvkům přidaným do G v předešlém kroku (viz řádky 14 a 15 algoritmu 1.6.1).

5. Jdi zpět na 1.

Algoritmus volby $B' \subseteq B$ se liší napříč jednotlivými variantami F4 a podrobněji se mu v této práci věnovat nebudeme.

Zbývá vysvětlit, jakým způsobem je v algoritmu 1.6.1 dosaženo bodu 3 ve výčtu výše. Tj. jaké polynomy do G doplnit, aby (1.3) platilo pro všechny dvojice prvků G odpovídající prvkům B'.

1.6.4 Množina L

V L jsou obsaženy prvky tvaru

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

pro nějaké $\{i, j\} \in B'$. Všimněme si, že se jedná přesně o jeden z polynomů, jejichž rozdíl tvoří polynom $\mathrm{S}(f_i, f_j)$ (viz definici 1.5.10).

1.6.5 Matice M, zobrazení π

Definice 1.6.2.

(i) Mějme polynom $f \in k[x_1, \ldots, x_n]$. Symbolem $\mathrm{Mon}(f)$ budeme značit množinu monomů obsažených v f. Tj. těch monomů, jimž v f přísluší nenulový koeficient z k.

(ii) Máme-li množinu polynomů $K \subseteq k[x_1, \ldots, x_n]$, pak položíme

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

Uvažujme neprázdnou množinu nenulových polynomů

$$H = \{f_1, \ldots, f_r\} \subseteq k[x_1, \ldots, x_n]$$

a monomiální uspořádání $>$. Pak

$$\mathrm{Mon}(H) = \{m_1, \ldots, m_s\}$$

pro nějaké $s \in \mathbb{N}$. Dále ať

$$X = \begin{pmatrix} m_{i_1} \\ \vdots \\ m_{i_s} \end{pmatrix}$$

je uspořádaná s-tice monomů z $\mathrm{Mon}(H)$ taková, že

$$m_{i_1} > \cdots > m_{i_s}.$$

Potom M je matice typu $r \times s$ taková, že

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

Neboli každý řádek matice M odpovídá jednomu polynomu z množiny H a v daném řádku jsou koeficienty příslušející jednotlivým monomům z množiny $\mathrm{Mon}(H)$ v pořadí určeném monomiálním uspořádáním $>$.

Dále zaveďme zobrazení π, které matici A typu $r \times s$ nad tělesem k přiřadí množinu polynomů reprezentovaných jejími řádky. Tj. pokud

$$AX = \begin{pmatrix} f_1 \\ \vdots \\ f_r \end{pmatrix}, \quad \text{pak} \quad \pi(A) = \{f_1, \ldots, f_r\}.$$

Příklad 1.6.3. Mějme množinu $H = \{f_1, \ldots, f_4\} \subseteq \mathbb{Z}_5[x, y, z]$ takovou, že

$$\begin{aligned} f_1 &= xz + 2z^2 + 2, \\ f_2 &= 3xy + 2xz, \\ f_3 &= 4z^2 + y + 1, \\ f_4 &= 2xz + 3z^2 + 2y, \end{aligned}$$

a uvažujme monomiální uspořádání $>_{grlex}$, kde $x > y > z$. Potom

$$\mathrm{Mon}(H) = \{xz, z^2, 1, xy, y\},$$

platí

$$xy >_{grlex} xz >_{grlex} z^2 >_{grlex} y >_{grlex} 1$$

a matice M je tvaru

$$M = \begin{pmatrix} 0 & 1 & 2 & 0 & 2 \\ 3 & 2 & 0 & 0 & 0 \\ 0 & 0 & 4 & 1 & 1 \\ 0 & 2 & 3 & 2 & 0 \end{pmatrix}.$$

Výše jsme popsali, jaký význam mají jednotlivé prvky matice M vzhledem k množině polynomů H, kterou M reprezentuje. Jaké polynomy jsou ale v množině H konkrétně obsaženy v rámci algoritmu 1.6.1?

Základem množiny H je množina L (viz sekci 1.6.4). Navíc jsou zde prvky tvaru $x^\alpha f_l$, kde $\alpha \in \mathbb{N}_0^n$ a $f_l \in G$ (viz sekci 1.6.2). Přesněji, množina H splňuje následující dvě podmínky.

(i) $L \subseteq H$.

(ii) Mějme libovolné $x^\beta \in \mathrm{Mon}(H)$. Potom pokud x^β je dělitelné monomem $\mathrm{LM}(f_l)$ pro nějaké $f_l \in G$, pak H obsahuje prvek $x^\alpha f_l$ takový, že $x^\beta = \mathrm{LM}(x^\alpha f_l)$.

Této podoby množiny M je dosaženo pomocí funkce, která je v pseudokódu algoritmu 1.6.1 značena jako *SpočtiM*. Ta je popsána v algoritmu 1.6.4.

Algoritmus 1.6.4 Funkce SpočtiM

Vstup: L, $G = (f_1, \ldots, f_t)$
Výstup: M
1: $H \leftarrow L$
2: *done* $\leftarrow \mathrm{LM}(H)$
3: **while** *done* $\neq \mathrm{Mon}(H)$ **do**
4: vyber největší $x^\beta \in (\mathrm{Mon}(H) \setminus \mathit{done})$ vzhledem k $>$
5: *done* $\leftarrow$ *done* $\cup \{x^\beta\}$
6: **if** existuje $f_l \in G$ takové, že $\mathrm{LM}(f_l) \mid x^\beta$ **then**
7: zvol libovolné $f_l \in G$ takové, že $\mathrm{LM}(f_l) \mid x^\beta$
8: $H \leftarrow H \cup \left\{ \dfrac{x^\beta}{\mathrm{LM}(f_l)} \cdot f_l \right\}$
9: $M \leftarrow$ matice koeficientů polynomů množiny H, jejíž sloupce odpovídají prvkům $\mathrm{Mon}(H)$ uspořádaným podle $>$
10: **return** M

Poznámka 1.6.5. V pseudokódu algoritmu 1.6.4 předpokládáme, že po aktualizaci množiny H je automaticky aktualizována i množina $\mathrm{Mon}(H)$, aniž by to bylo explicitně uvedeno.

Podmínku (i) splní algoritmus 1.6.4 hned úvodním přiřazením. A vzhledem k tomu, že množina H je ve zbytku běhu algoritmu už jen rozšiřována, tato podmínka již nemůže být porušena.

Dále algoritmus 1.6.4 pracuje s množinou *done* $\subseteq \mathrm{Mon}(H)$, v níž si udržuje ty monomy, pro něž již byla zajištěna podmínka (ii). I množina *done* je po dobu běhu algoritmu jen rozšiřována. Toto je validní přístup, jelikož pro daný monom $x^\beta \in \mathrm{Mon}(H)$ mohou nastat jen dvě možnosti.

(i) Neexistuje $f_l \in G$ takové, že $\mathrm{LM}(f_l) \mid x^\beta$ a podmínka (ii) je triviálně splněna.

(ii) Existuje $f_l \in G$ takové, že $\mathrm{LM}(f_l) \mid x^\beta$, do H je přidáno vhodné $x^\alpha f_l$ a jelikož z H po celou dobu běhu algoritmu žádné prvky neodstraňujeme, podmínka (ii) bude pro x^β splněna vždy.

Úvodní přiřazení *done* $\leftarrow \mathrm{LM}(H)$ nastává v okamžiku, kdy $H = L$, a tedy $\mathrm{LM}(H) = \mathrm{LM}(L)$. Zde připomeňme, že pro $\{i, j\} \in B'$ jsou v L (a tedy i v H) oba polynomy

$$p = \frac{\mathrm{lcm}(\mathrm{LM}(f_i), \mathrm{LM}(f_j))}{\mathrm{LT}(f_i)} \cdot f_i \quad \text{i} \quad q = \frac{\mathrm{lcm}(\mathrm{LM}(f_i), \mathrm{LM}(f_j))}{\mathrm{LT}(f_j)} \cdot f_j.$$

Zjevně platí $\mathrm{LM}(f_j) \mid \mathrm{LM}(p)$, ale v L (a tedy i v H) už je zahrnut polynom q, který splňuje

$$q = cx^\alpha f_j \quad \text{pro nějaké} \quad c \in k, \alpha \in \mathbb{N}_0^n$$

a zároveň

$$\mathrm{LM}(q) = \mathrm{LM}(p).$$

Podmínka (ii) je tedy splněna pro $\mathrm{LM}(p)$ a naprosto analogicky i pro $\mathrm{LM}(q)$ a celé $\mathrm{LM}(H)$. Proto můžeme přiřazení *done* $\leftarrow \mathrm{LM}(H)$ provést hned v úvodu.

Závěrem této sekce ukažme, že algoritmus 1.6.4 skončí po konečném počtu kroků. Všimněme si, že jakmile je ve *while* cyklu zvoleno nějaké $x^\beta \in (\mathrm{Mon}(H) \setminus \mathit{done})$, v žádném dalším průchodu cyklem se jím již zabývat nebudeme, jelikož hned v následujícím kroku je toto x^β přidáno do množiny *done*.

Zároveň jediné monomy, které přibudou do množiny $\mathrm{Mon}(H)$ v jednom průchodu cyklem, jsou ty z množiny $\mathrm{Mon}\left(\frac{x^\beta}{\mathrm{LM}(f_l)} \cdot f_l\right)$ pro $f_l \in G$ takové, že $\mathrm{LM}(f_l) \mid x^\beta$. Zřejmě platí

$$\mathrm{LM}\left(\frac{x^\beta}{\mathrm{LM}(f_l)} \cdot f_l\right) = x^\beta,$$

ale x^β již v $\mathrm{Mon}(H)$ bylo obsaženo. Tudíž všechny monomy, které v tomto průchodu cyklem přibudou v množině $\mathrm{Mon}(H)$, jsou vzhledem ke zvolenému monomiálnímu uspořádání $>$ ostře menší než x^β.

Z toho dále plyne, že monomy x^β volené v jednotlivých průchodech cyklem tvoří vzhledem k uspořádání $>$ klesající posloupnost. Monomiální uspořádání $>$ je dobré uspořádání (viz definici 1.2.3), a tedy každá neprázdná množina monomů má vůči němu nejmenší prvek (viz definici 1.2.2). Posloupnost monomů x^β nemůže klesat dále než k tomuto nejmenšímu prvku a musí být konečná. Všechny prvky $\mathrm{Mon}(H)$ budou proto v konečném počtu kroků přidány do množiny *done* a algoritmus skončí.

1.6.6 Matice N

Definice 1.6.6–1.6.9 jsou převzaty z [4] a [5].

Definice 1.6.6. Mějme matici A a její prvek a_{ij} (tj. prvek v i-tém řádku a j-tém sloupci). Řekneme, že a_{ij} je *pivot*, pokud a_{ij} je zleva první nenulový prvek v i-tém řádku.

Definice 1.6.7. Řekneme, že matice A je v *odstupňovaném tvaru*, pokud

(i) každý nulový řádek matice A leží pod všemi nenulovými řádky matice A,

(ii) každý pivot matice A je více vpravo než všechny pivoty řádků nad ním.

Definice 1.6.8. Řekneme, že matice A je v *redukovaném odstupňovaném tvaru*, pokud je v odstupňovaném tvaru a navíc

(i) každý prvek matice A, který leží ve sloupci nad některým pivotem, je roven 0,

(ii) každý pivot matice A je roven 1.

Definice 1.6.9. Mějme matici A. Následující úpravy matice A nazveme *elementárními řádkovými úpravami*.

(i) Prohození dvou řádků matice A,

(ii) vynásobení řádku matice A nenulovou konstantou,

(iii) přičtení nenulového násobku řádku matice A k jinému jejímu řádku.

Poznámka 1.6.10. Jak již bylo naznačeno v pseudokódu algoritmu 1.6.4, matice N

(i) je v redukovaném odstupňovaném tvaru (viz definici 1.6.8),

(ii) vznikla z matice M (viz sekci 1.6.5) pomocí elementárních řádkových úprav (viz definici 1.6.9).

Poznámka 1.6.11. Interpretace matice N je v zásadě stejná jako ta matice M. Její sloupce odpovídají jednotlivým monomům z $\mathrm{Mon}(H)$. Každý řádek reprezentuje jeden polynom, přičemž prvky daného řádku jsou koeficienty u příslušných monomů reprezentovaného polynomu (viz sekci 1.6.5).

Poznámka 1.6.12. Všimněme si, že pokud polynomy odpovídající řádkům matice M jsou prvky ideálu $I \subseteq k[x_1, \ldots, x_n]$, pak polynomy odpovídající řádkům matice N jsou prvky téhož ideálu. To vyplývá z bodu (ii) poznámky 1.6.10 a z definice 1.6.9.

Odsud pak dále můžeme vidět, že přidáním polynomů odpovídajících řádkům matice N do původní báze F ideálu I nezměníme ideál, který báze F generuje.

1.6.7 Množina N^+

Uvažujme množinu M', která obsahuje polynomy reprezentované řádky matice M, a množinu N', která obsahuje polynomy reprezentované řádky matice N. Pro množinu N^+ platí

$$N^+ = \{f \in N' \mid \mathrm{LM}(f) \notin \langle \mathrm{LM}(M') \rangle\}.$$

Poznámka 1.6.13. Máme-li $f \in N'$ a chceme-li rozhodnout, jestli $f \in N^+$, stačí ověřit, jestli existuje $g \in M'$ takové, že $\mathrm{LM}(g) \mid \mathrm{LM}(f)$ (viz lemma 1.4.6). Přičemž ověření dělitelnosti dvou monomů je triviální.

1.6.8 Správnost algoritmu 1.6.1 (F4)

Všimněme si, že každá dvojice $\{i, j\}$ taková, že $1 \leq i < j \leq t$, je v nějakém okamžiku obsažena v množině B a někdy později je z ní odebrána. V příslušné iteraci *while* cyklu jsou v množině L obsaženy polynomy

$$\frac{\mathrm{lcm}(\mathrm{LM}(f_i), \mathrm{LM}(f_j))}{\mathrm{LT}(f_i)} \cdot f_i \quad \text{a} \quad \frac{\mathrm{lcm}(\mathrm{LM}(f_i), \mathrm{LM}(f_j))}{\mathrm{LT}(f_j)} \cdot f_j.$$

Matice M potom obsahuje řádky odpovídající těmto dvěma polynomům. Polynom $\mathrm{S}(f_i, f_j)$ je rozdílem těchto dvou polynomů, a je tedy obsažen v lineárním prostoru generovaném polynomy odpovídajícími řádkům matice M. Matice N, jakožto redukovaný odstupňovaný tvar matice M, sestává z řádků odpovídajících polynomům tvořícím bázi tohoto lineárního prostoru. $\mathrm{S}(f_i, f_j)$ je proto lineární kombinací polynomů odpovídajících řádkům matice N.

Toto vyjádření polynomu $\mathrm{S}(f_i, f_j)$ nám poskytuje jeho *standardní reprezentaci*

$$\mathrm{S}(f_i, f_j) = A_1 g_1 + \cdots + A_t g_t$$

(viz definici 1.5.14), kde $\forall i \in \{1, \ldots, t\} : g_i \in G$, přičemž proměnnou t i množinu G uvažujeme po provedení celého *for* cyklu. Z existence standardní reprezentace už přímo plyne, že po dokončení algoritmu 1.6.1 množina G tvoří hledanou Gröbnerovu bázi (viz větu 1.5.15).

Další podrobnosti k algoritmu 1.6.1 včetně důkazu konečnosti jsou k nalezení v [2].

1.7 Řešení soustavy polynomiálních rovnic

V předchozích sekcích jsme ukázali, co jsou Gröbnerovy báze a jak je lze spočítat. Zbývá ukázat, jakým způsobem je můžeme využít pro řešení soustavy polynomiálních rovnic.

Uvažujme soustavu

$$\begin{aligned} f_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ f_s(x_1, \ldots, x_n) &= 0, \end{aligned} \tag{1.4}$$

kde $f_1, \ldots, f_s \in k[x_1, \ldots x_n]$, a Gröbnerovu bázi G ideálu $\langle f_1, \ldots, f_s \rangle$. Potom soustava

$$\begin{aligned} g_1(x_1, \ldots, x_n) &= 0 \\ &\vdots \\ g_t(x_1, \ldots, x_n) &= 0, \end{aligned} \tag{1.5}$$

má identickou množinu řešení jako soustava (1.4) (viz poznámku 1.4.16 a větu 1.5.3).

Navíc pokud G je *redukovaná* Gröbnerova báze $\langle f_1, \ldots, f_s \rangle$, pak existuje $i \in \{1, \ldots, t\}$ takové, že g_i obsahuje jen jednu neznámou x_l. Pro tuto neznámou pak lze rychle určit množinu možných řešení. Dále bude existovat $j \in \{1, \ldots, t\}$ takové, že g_j obsahuje pouze proměnné x_l a x_k. Proměnnou x_l již známe, můžeme ji tedy dosadit a získat řešení x_k. Tímto způsobem je možné vyřešit celou soustavu (1.5), a tedy i původní soustavu (1.4) [3].

Celý postup řešení soustavy polynomiálních rovnic pomocí Gröbnerových bází tak můžeme shrnout do několika bodů.

1. Zadanou soustavu si zapíšeme ve tvaru (1.4). Tím získáme polynomy $f_1, \ldots, f_s$.
2. Spočteme některou Gröbnerovu bázi G ideálu $\langle f_1, \ldots, f_s \rangle$.
3. Gröbnerovu bázi G upravíme na redukovanou Gröbnerovu bázi G'.
4. Vyřešíme soustavu polynomiálních rovnic danou polynomy z G' pomocí dosazovací metody popsané v předešlém odstavci.

Kapitola 2

LFSR

Předmětem této práce je kryptoanalýza proudových šifer založených na LFSR (linear feedback shift register – posuvný registr s lineární zpětnou vazbou). Podívejme se tedy na to, co přesně LFSR je a jakým způsobem ho lze pro konstrukci proudových šifer využít. Závěrem této kapitoly také věnujme pozornost způsobu, jakým lze takto vytvořenou proudovou šifru vyjádřit jako systém polynomiálních rovnic.

Nebude-li uvedeno jinak, obsah této kapitoly bude vycházet z [1].

Definice 2.0.1. Mějme těleso $\mathbb{F}_q$ a $L \in \mathbb{N}$.

(i) *LFSR* délky L nad $\mathbb{F}_q$ je konečný automat, jehož výstupem je nekonečná posloupnost $s = (s_t)_{t \geq 0}$ prvků $\mathbb{F}_q$ daná rekurentním vztahem

$$s_{t+L} = \sum_{i=1}^{L} c_i s_{t+L-i}, \quad \forall t \geq 0. \tag{2.1}$$

(ii) Koeficienty $c_1, \ldots, c_L$ jsou také prvky $\mathbb{F}_q$ a nazýváme je *koeficienty zpětné vazby* LFSR.

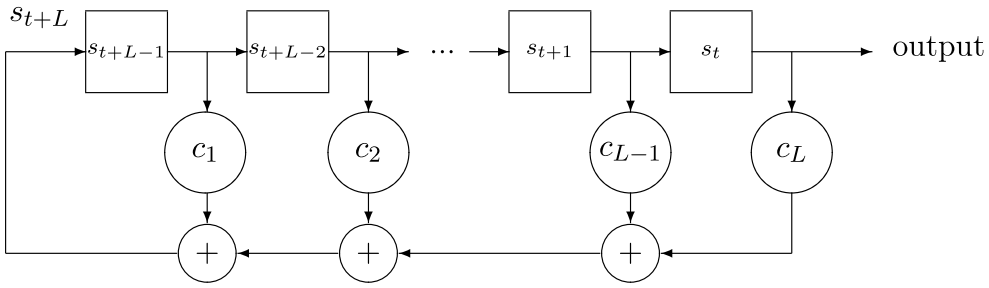
(iii) Hodnoty $s_t, \ldots, s_{t+L-1}$ udávají aktuální *stav* LFSR. Přičemž LFSR je inicializován *počátečním stavem*, tj. hodnotami $s_0, \ldots, s_{L-1}$.

Poznámka 2.0.2. Výstup každého LFSR je jednoznačně určen jeho koeficienty zpětné vazby a počátečním stavem.

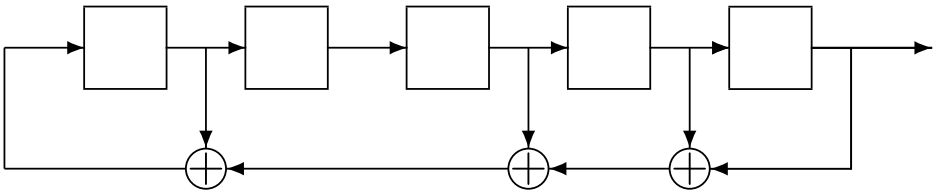
Definice 2.0.3. Mějme LFSR délky L s koeficienty zpětné vazby $c_1, \ldots, c_L$.

(i) Tyto koeficienty typicky reprezentuje *polynom zpětné vazby* daný vztahem

$$P(x) = 1 - \sum_{i=1}^{L} c_i x^i.$$



Obrázek 2.1: Schéma LFSR [1]



Obrázek 2.2: Binární LFSR s koeficienty $(c_1, c_2, c_3, c_4, c_5) = (1, 0, 1, 1, 1)$

(ii) Alternativně jsou koeficienty zpětné vazby vyjádřeny *charakteristickým polynomem*

$$P^*(x) = x^L P(x^{-1}) = x^L - \sum_{i=1}^{L} c_i x^{L-i}.$$

Příklad 2.0.4. Uvažujme binární LFSR z obrázku 2.2. Pro něj platí

$$P(x) = 1 + x + x^3 + x^4 + x^5 \quad a \quad P^*(x) = 1 + x + x^2 + x^3 + x^5.$$

Pevně daný keystream může být výstupem vícero různých LFSR. Se snahou o co nejjednodušší implementaci generátoru keystreamu přirozeně přichází otázka, jak lze charakterizovat „minimální“ generátor, jehož výstupem je daný keystream.

Tvrzení 2.0.5. *Mějme posloupnost $(s_t)_{t\geq 0}$ danou lineárním rekurentním vztahem. Pak existuje jednoznačně určený polynom P_0 s absolutním členem rovným* 1 *takový, že členy posloupnosti $(s_t)_{t\geq 0}$ jsou dány rovností*

$$\sum_{t\geq 0} s_t x^t = \frac{Q_0(x)}{P_0(x)},$$

kde polynomy P_0 a Q_0 jsou vzájemně nesoudělné.

Nejkratší LFSR, který generuje posloupnost $(s_t)_{t\geq 0}$, je délky

$$L = \max(\deg(P_0), \deg(Q_0) + 1)$$

a jeho polynom zpětné vazby je roven P_0. Polynom $P_0^* = x^L P_0(x^{-1})$ je charakteristickým polynomem tohoto nejkratšího LFSR.

Definice 2.0.6. Charakteristický polynom P_0^* z tvrzení 2.0.5 nazveme *minimální polynom* dané posloupnosti.

Tvrzení 2.0.7. *Perioda posloupnosti dané lineárním rekurentním vztahem je rovna nejmenšímu $e \in \mathbb{N}$ takovému, že $P_0(x) \mid x^e + 1$.*

Následující definice je převzata z [6].

Definice 2.0.8.

(i) Mějme $\alpha \in \mathbb{F}_{q^d}$. Řekneme, že α je *primitivní prvek*, pokud α generuje $\mathbb{F}_{q^d}^*$.

(ii) Ať $f(x) \in \mathbb{F}_q[x]$ je ireducibilní polynom stupně d takový, že

$$\mathbb{F}_{q^d} = \mathbb{F}_q[x]/f(x),$$

a ať α je kořenem $f(x)$. Řekneme, že $f(x)$ je *primitivní polynom*, pokud α je primitivní prvek.

Tvrzení 2.0.9. *Posloupnost daná lineárním rekurentním vztahem dosahuje maximální možné periody, pokud její polynom zpětné vazby je primitivní polynom.*

V následujících sekcích budeme uvažovat pouze LFSR nad $\mathbb{F}_2$.

2.1 Konstrukce proudových šifer

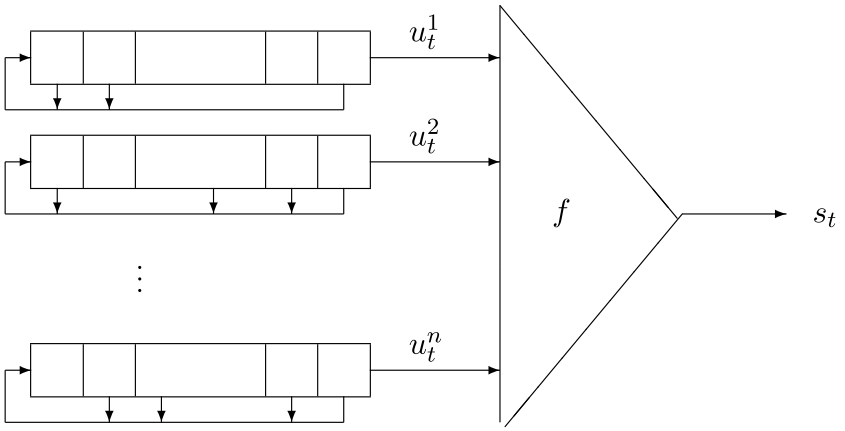
Samotný výstup jednoho LFSR není jako zdroj keystreamu proudové šifry postačující kvůli pevně daným lineárním závislostem mezi jeho jednotlivými bity. Proudovou šifru nicméně pomocí LFSR zkonstruovat lze. Ukážeme si jakými způsoby.

2.1.1 Kombinační generátor

Kombinační generátor je generátor keystreamu proudové šifry, který vznikne složením několika LFSR. Výstupy jednotlivých LFSR jsou kombinovány boolovskou funkcí. Posloupnost $(s_t)_{t\geq 0}$, která je výstupem kombinačního generátoru, je tedy dána vztahem

$$s_t = f(u_t^1, \ldots, u_t^n), \quad \forall t \geq 0, \tag{2.2}$$

kde $(u_t^i)_{t\geq 0}$ značí posloupnost generovanou i-tým LFSR a kde f je funkcí n proměnných. V případě, kdy je kombinační generátor sestaven z n LFSR nad $\mathbb{F}_q$, je tedy f funkcí z $\mathbb{F}_q^n$ do $\mathbb{F}_q$. Schéma kombinačního generátoru je zachyceno na obrázku 2.3.



Obrázek 2.3: Kombinační generátor [1]

Funkce f by měla být nelineární (jinak by celá konstrukce postrádala smysl) a *vyvážená*, tj. její výstup by se měl blížit rovnoměrnému rozdělení. Pro jednotlivá LFSR by měl být volen polynom zpětné vazby, který je primitivním polynomem. Volba primitivního polynomu má pozitivní vliv jak na délku periody výstupní posloupnosti, tak na její statistické vlastnosti.

Parametry jednotlivých LFSR a funkce f jsou typicky veřejné. Tajným parametrem jsou počáteční stavy jednotlivých LFSR, které jsou odvozeny z tajného klíče šifry.

2.1.2 Filtr generátor

Na rozdíl od kombinačního generátoru filtr generátor sestává z jediného LFSR, jehož stav je filtrován nelineární funkcí. Přesněji řečeno výstupní posloupnost filtr generátoru odpovídá výstupu nelineární funkce, která na vstupu přijímá část stavu LFSR.

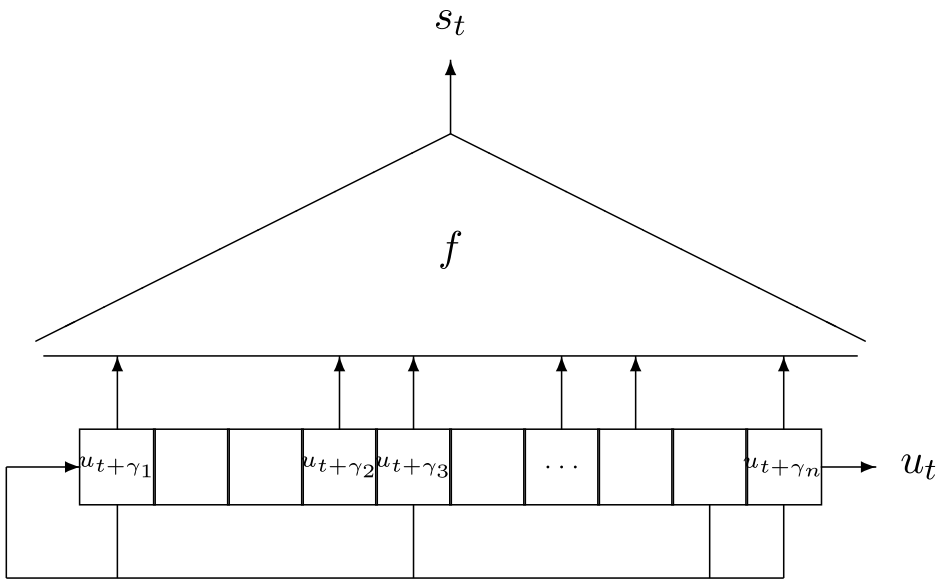
Označme $(u_t)_{t\geq 0}$ posloupnost generovanou LFSR délky L. Výstupní posloupnost filtr generátoru $(s_t)_{t\geq 0}$ je pak dána vztahem

$$s_t = f(u_{t+\gamma_1}, \ldots, u_{t+\gamma_n}), \quad \forall t \geq 0, \tag{2.3}$$

kde f je funkcí n proměnných, $n \leq L$, a $(\gamma_i)_{1\leq i\leq n}$ je klesající posloupnost nezáporných celých čísel. Pro lepší představu viz obrázek 2.4.

Stejně jako u kombinačního generátoru by funkce f měla být vyvážená a pro jednotlivá LFSR by měl být volen polynom zpětné vazby, který je primitivním polynomem.

Polynom zpětné vazby, funkce f a posloupnost $(\gamma_i)_{1\leq i\leq n}$ jsou veřejnými parametry filtr generátoru. Tajným parametrem je počáteční stav LFSR, který je odvozen z tajného klíče šifry.



Obrázek 2.4: Filtr generátor [1]

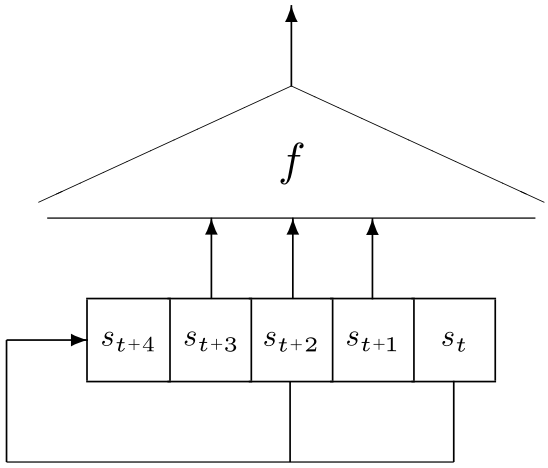
2.2 Převod proudové šifry na soustavu rovnic

Předpokládejme, že známe dvojici *otevřený text – šifrový text*, kde šifrový text byl z otevřeného textu získán pomocí proudové šifry, jejíž generátor keystreamu je implementován jedním ze dvou způsobů popsaných výše (viz sekci 2.1). Operací *xor* po prvcích aplikovanou na posloupnosti bitů otevřeného a šifrového textu získáme keystream, který byl k šifrování použit.

Podíváme-li se na vztahy (2.2) a (2.3), znalost keystreamu odpovídá znalosti hodnot s_t pro t odpovídající délce použitého keystreamu. Dosazením jednoho z těchto dvou vztahů do rovnice (2.1) získáme polynomiální rovnici, která dává do souvislosti tajné (!) počáteční stavy jednotlivých LFSR použitých v generátoru keystreamu a známou hodnotu jednoho z bitů keystreamu.

Označme n celkový počet bitů počátečních stavů všech LFSR obsažených v generátoru keystreamu. Zkonstruujeme-li výše popsaným způsobem rovnice odpovídající prvním n bitům keystreamu, získáme soustavu n polynomiálních rovnic nad $\mathbb{F}_2$, jejímž řešením je hodnota počátečních stavů všech použitých LFSR.

Poznámka 2.2.1. Teoreticky může nastat situace, kdy se v soustavě rovnic odpovídající prvním n bitům keystreamu neobjeví proměnná, která značí nějaký bit jednoho z počátečních stavů použitých LFSR. Ani po vyřešení soustavy pak neznáme tuto část počátečního stavu. To ovšem z pohledu prolomení šifry nepředstavuje problém. Pokud daným bitem počátečního stavu nebyl nijak ovlivněn žádný z prvních n bitů keystreamu, nebude jím ovlivněn ani žádný další a jeho znalost není podstatná.



Obrázek 2.5: Filtr generátor z příkladu 2.2.3

Poznámka 2.2.2. Pro odhalení počátečních stavů použitých LFSR není nutné, aby soustava polynomiálních rovnic byla sestavena na základě *prvních* n bitů keystreamu. V praxi většinou všechny tyto bity nejsou k dispozici. Prvních m vygenerovaných bitů je typicky zahozeno a keystream je tvořem až bity následujícími. Přičemž hodnota m patří mezi veřejně známé parametry šifry.

Sestavíme-li soustavu polynomiálních rovnic na základě bitů $s_m, \ldots, s_{m+n-1}$, získáme jejím vyřešením vnitřní stavy jednotlivých LFSR po prvních m taktech. Pak stačí provést m taktů jednotlivých LFSR ve zpětném chodu, což už není nijak výpočetně náročné. Vnitřní stav jednotlivých LFSR po m zpětných taktech je roven hledanému počátečnímu stavu.

Metoda efektivního řešení soustavy polynomiálních rovnic nad konečným tělesem ve více proměnných je popsána v kapitole 1.

Příklad 2.2.3. Uvažme filtr generátor délky $L = 5$. Jeho počáteční stav označme $(s_4, s_3, s_2, s_1, s_0)$. Koeficienty zpětné vazby ať jsou

$$(c_1, c_2, c_3, c_4, c_5) = (0, 0, 1, 0, 1)$$

a funkce f ať je dána předpisem

$$f(s_{t+3}, s_{t+2}, s_{t+1}) = s_{t+1}s_{t+2} + s_{t+3}. \tag{2.4}$$

Značení zavádíme v souladu s definicí 2.0.1 a obrázkem 2.1. Pro lepší představu viz obrázek 2.5.

V tabulce 2.1 je zachycen stav LFSR v prvních pěti taktech.

Předpokládejme, že keystream produkovaný generátorem výše začíná posloupností $(k_0, k_1, k_2, k_3, k_4)$. Z rovnice (2.4), ze stavů zachycených v tabulce 2.1 a z posloupnosti $(k_0, k_1, k_2, k_3, k_4)$ pak získáme rovnice

Takt	Stav				
0	s_4	s_3	s_2	s_1	s_0
1	$s_0 + s_2$	s_4	s_3	s_2	s_1
2	$s_1 + s_3$	$s_0 + s_2$	s_4	s_3	s_2
3	$s_2 + s_4$	$s_1 + s_3$	$s_0 + s_2$	s_4	s_3
4	$s_0 + s_2 + s_3$	$s_2 + s_4$	$s_1 + s_3$	$s_0 + s_2$	s_4

Tabulka 2.1: Stavy LFSR z příkladu 2.2.3

$$\begin{aligned} s_1s_2 + s_3 &= k_0, \\ s_2s_3 + s_4 &= k_1, \\ s_3s_4 + s_0 + s_2 &= k_2, \\ s_4(s_0 + s_2) + s_1 + s_3 &= k_3, \\ (s_0 + s_2)(s_1 + s_3) + s_2 + s_4 &= k_4. \end{aligned} \tag{2.5}$$

Uvažme konkrétně případ $(k_0, k_1, k_2, k_3, k_4) = (0, 1, 0, 0, 0)$. Pak na základě soustavy 2.5 můžeme psát

$$\begin{aligned} s_1s_2 + s_3 &= 0, \\ s_2s_3 + s_4 + 1 &= 0, \\ s_3s_4 + s_0 + s_2 &= 0, \\ s_4(s_0 + s_2) + s_1 + s_3 &= 0, \\ (s_0 + s_2)(s_1 + s_3) + s_2 + s_4 &= 0. \end{aligned} \tag{2.6}$$

Využitím algoritmu F4 pro výpočet Gröbnerových bází (viz kapitolu 1) získáme ekvivalentní soustavu

$$\begin{aligned} s_0 + 1 &= 0, \\ s_1 &= 0, \\ s_2 + 1 &= 0, \\ s_3 &= 0, \\ s_4 + 1 &= 0, \end{aligned} \tag{2.7}$$

ze které je již přímo vidět řešení $(s_0, s_1, s_2, s_3, s_4) = (1, 0, 1, 0, 1)$. Toto řešení opravdu odpovídá počátečnímu stavu filtr generátoru popsaného v úvodu příkladu, jehož výstupem byl keystream $(0, 1, 0, 0, 0)$.

Kapitola 3

Experimenty

V předchozích kapitolách jsme popsali, jak proudovou šifru založenou na LFSR převést na soustavu polynomiálních rovnic a jak tuto soustavu efektivně vyřešit. Vhodným použitím metody *guess-and-determine* lze řešení získané soustavy polynomiálních rovnic dále zefektivnit.

Obsahem této kapitoly bude popis a výsledky měření, jejichž primárním cílem je právě nalezení vhodného způsobu použití metody *guess-and-determine* pro řešení soustavy polynomiálních rovnic reprezentující proudovou šifru založenou na LFSR.

3.1 Hardware

Všechna měření byla provedena na stroji, mezi jehož hlavní specifika patří

- dva procesory Intel Xeon Gold 6136 o 12 jádrech a frekvenci 3,0 GHz,
- 755 GB RAM.

3.2 Metody výpočtu

V obecné rovině jsme při výpočtech využili dva různé postupy.

3.2.1 Referenční výpočet

Jako referenční označujeme přímočarý výpočet bez použití metody *guess-and-determine*. Tento postup má význam zejména pro vyhodnocení kvality ostatních postupů založených na *guess-and-determine*.

V rámci referenčního výpočtu nejprve pro daný keystream a parametry generátoru keystreamu sestavíme soustavu polynomiálních rovnic (viz sekci 2.2). Tu následně vyřešíme způsobem blíže rozebraným v sekci 1.7.

3.2.2 Guess-and-determine

Zopakujme, že v rámci úlohy, kterou řešíme, máme dány parametry generátoru keystreamu a vyprodukovaný keystream. Cílem je najít počáteční stav generátoru keystreamu.

Metoda guess-and-determine je postavena na rozkladu na vícero jednodušších instancí stejného problému.

V rámci této metody je nejprve zapotřebí zvolit množinu neznámých obsaženou v hledaném řešení. V našem případě se může jednat o podmnožinu bitů počátečního stavu generátoru keystreamu nebo obecněji o množinu aritmetických výrazů závislých na bitech počátečního stavu generátoru keystreamu. Při této volbě se snažíme využít nějaké nedokonalosti ve struktuře šifry.

Následně uvažujeme všechny možné kombinace hodnot, kterých neznámé ze zvolené množiny mohou nabývat. Každá z těchto kombinací vede na dílčí podproblém, který musíme vyřešit. V rámci tohoto podproblému předpokládáme, že zvolená množina neznámých opravdu nabývá uvažované kombinace hodnot. To nám při vhodně zvolené množině neznámých značně usnadní řešení daného podproblému.

Vzhledem k tomu, že v řadě takto získaných podproblémů pracujeme s mylným předpokladem, je zapotřebí všechna získaná řešení podrobit nějaké formě zkoušky. Jen ta řešení dílčích podproblémů, u nichž zkouška potvrdí správnost ve smyslu řešení původního problému, jsou zahrnuta do výsledného řešení.

3.2.2.1 Zkouška

Chceme-li ověřit, že nalezený počáteční stav je správný, stačí simulovat běh generátoru keystreamu o daných parametrech a nalezeném počátečním stavu a porovnat keystream, který produkuje, s tím zadaným. Přesně takový postup jsme použili v měřeních obsažených v této práci.

Při kontrole shodnosti dvou keystreamů se nabízí otázka, kolik jejich bitů budeme porovnávat. V rámci referenčního výpočtu do soustavy zahrnujeme tolik rovnic, kolik je bitů vnitřního stavu generátoru keystreamu. Pro generátor o n bitech vnitřního stavu tak platí, že řešení získaná referenčním výpočtem se se zadaným keystreamem shodují na prvních n bitech. Stejnou „přesnost“ řešení tedy budeme požadovat i od řešení získaných metodou guess-and-determine. V rámci zkoušky tedy kontrolujeme tolik bitů keystreamu, kolik je bitů vnitřního stavu generátoru.

Příklad 3.2.1. Uvažme znovu filtr generátor z příkladu 2.2.3. Jako neznámou, jejíž hodnotu budeme „hádat“, zvolme bit s_0.

Nejprve předpokládejme $s_0 = 0$. Za tohoto předpokladu dostaneme na-

místo soustavy (2.6) soustavu

$$\begin{aligned} s_1 s_2 + s_3 &= 0, \\ s_2 s_3 + s_4 + 1 &= 0, \\ s_3 s_4 + s_2 &= 0, \\ s_4 s_2 + s_1 + s_3 &= 0, \\ s_2(s_1 + s_3) + s_2 + s_4 &= 0. \end{aligned}$$

Tato soustava nemá nad $\mathbb{F}_2$ žádné řešení. Není tedy třeba provádět zkoušku.

Dále předpokládejme $s_0 = 1$. Nyní získáme soustavu

$$\begin{aligned} s_1 s_2 + s_3 &= 0, \\ s_2 s_3 + s_4 + 1 &= 0, \\ s_3 s_4 + 1 + s_2 &= 0, \\ s_4(1 + s_2) + s_1 + s_3 &= 0, \\ (1 + s_2)(s_1 + s_3) + s_2 + s_4 &= 0. \end{aligned}$$

Tato soustava má jediné řešení $(s_1, s_2, s_3, s_4) = (0, 1, 0, 1)$. Pokud tedy doplníme předpoklad $s_0 = 1$, máme $(s_0, s_1, s_2, s_3, s_4) = (1, 0, 1, 0, 1)$. Následně ověříme, že zadaný filtr generátor s tímto počátečním stavem opravdu vyprodukuje zadaný keystream, a zařadíme toto dílčí řešení do výsledné množiny řešení.

Jelikož jsme již vyzkoušeli všechny možné hodnoty neznámé s_0, výpočet ukončíme a vracíme (jediné získané) řešení $(s_0, s_1, s_2, s_3, s_4) = (1, 0, 1, 0, 1)$.

3.3 Měření času

Tím, na co se v této práci soustředíme, je nalezení co možná nejrychlejšího způsobu řešení soustavy polynomiálních rovnic. Podívejme se tedy také na to, jakým způsobem čas měříme.

Předně uveďme, že veškeré výpočty, které provádíme, jsou spuštěny jako proces o jednom vláknu. Nedochází k žádné paralelizaci.

3.3.1 Referenční výpočet

V případě referenčního výpočtu je měření času jednoduché. Zadanou soustavu polynomiálních rovnic řešíme přímo, takže jako výsledný čas bereme čas jejího řešení.

3.3.2 Guess-and-determine

U guess-and-determine je situace o něco složitější. Probíhá zde několik výpočtů dílčích soustav polynomiálních rovnic a pro každý z nich ještě kontrola řešení. Výsledný čas je tedy součtem časů řešení dílčích soustav a časů kontroly dílčích řešení.

3.3.3 Generování rovnic

Čas potřebný pro vygenerování rovnic na základě parametrů generátoru keystreamu do výsledného času započten není.

To odpovídá reálné situaci. Parametry generátoru keystreamu jsou veřejné (viz sekci 2.1), a útočník tedy má možnost si rovnice odpovídající jednotlivým bitům keystreamu vygenerovat předem. V okamžiku získání použitého keystreamu mu pak stačí každou z vygenerovaných rovnic doplnit o příslušný bit keystreamu, což je operace, která v porovnání s ostatními vyžaduje zanedbatelné množství času.

3.4 Průběh měření

Nebude-li řečeno jinak, bude postup jednotlivých měření v této kapitole vždy stejný.

1. Zvolíme parametry generátoru keystreamu.
2. Vygenerujeme 30 náhodných keystreamů stejné bitové délky, jakou má vnitřní stav generátoru.
3. Různými způsoby (viz sekce 3.6.1-3.6.3, 3.7.1-3.7.4 a 3.8.1-3.8.3) zkoušíme řešit soustavy polynomiálních rovnic nad $\mathbb{F}_2$ získané z parametrů generátoru a jednotlivých keystreamů. Při tom zaznamenáváme čas způsobem popsaným v sekci 3.3.

Typickým výstupem pak je aritmetický průměr a směrodatná odchylka časů (v sekundách) potřebných pro řešení jednotlivých soustav polynomiálních rovnic daným způsobem.

Poznámka 3.4.1. V bodu 1 postupu výše vždy pro zvolené bitové délky jednotlivých LFSR generátoru keystreamu volíme takový polynom zpětné vazby, který je primitivním polynomem. Postupujeme tedy v souladu s doporučením plynoucím z tvrzení 2.0.9.

3.5 Monomiální uspořádání

V sekci 1.2 jsme představili několik různých monomiálních uspořádání. Konkrétní volba monomiálního uspořádání může mít značný vliv na dobu běhu algoritmu F4, a tedy i na celkový čas potřebný pro řešení soustavy polynomiálních rovnic. Pro řadu výpočtů je nejefektivnější volbou uspořádání $>_{grevlex}$ [2].

uspořádání	aritmetický průměr [s]	směrodatná odchylka [s]
$>_{lex}$	2589.461	6649.533
$>_{grlex}$	0.134	0.088
$>_{grevlex}$	0.132	0.090

Tabulka 3.1: Monomiální uspořádání na 28bitovém kombinačním generátoru

uspořádání	aritmetický průměr [s]	směrodatná odchylka [s]
$>_{grlex}$	3367.480	811.352
$>_{grevlex}$	817.260	50.134

Tabulka 3.2: Mnomiální uspořádání na 64bitovém kombinačním generátoru

3.5.1 Experimentální porovnání

Uvažme kombinační generátor sestávající ze 4 LFSR o 7 bitech vnitřního stavu. Koeficienty zpětné vazby každého z nich jsou

$$(c_1, c_2, c_3, c_4, c_5, c_6, c_7) = (1, 0, 0, 0, 0, 0, 1).$$

Označme u_t^i t-tý výstupní bit i-tého LFSR. Pro t-tý výstupní bit generátoru platí

$$s_t = u_t^1 + u_t^2 u_t^3 + u_t^1 u_t^2 u_t^3 + u_t^1 u_t^3 u_t^4. \tag{3.1}$$

Tomuto vztahu se budeme blíže věnovat v následujících sekcích této kapitoly. Zatím ho vezměme jako daný.

Porovnejme výkonnost referenčního výpočtu (viz sekci 3.3.1) při použití monomiálních uspořádání $>_{lex}$, $>_{grlex}$ a $>_{grevlex}$. Měření proběhlo v souladu se sekcí 3.4 a jeho výsledky vidíme v tabulce 3.1.

Můžeme říct, že uspořádání $>_{lex}$ je jednoznačně nejpomalejší a při jeho použití se časy výpočtu jednotlivých instancí z datové sady velmi výrazně liší. Uspořádání $>_{grlex}$ a $>_{grevlex}$ z tohoto porovnání vycházejí velmi podobně. Zkusme tedy problém přeškálovat.

Vezměme kombinační generátor sestávající ze 4 LFSR o 16 bitech vnitřního stavu. Koeficienty zpětné vazby každého z nich jsou

$$(c_1, \ldots, c_{16}) = (0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1).$$

Pro kombinaci výstupních bitů jednotlivých LFSR opět použijme vztah (3.1).

Nyní porovnejme referenční výpočty již pouze za využití $>_{grlex}$ a $>_{grevlex}$. Měření proběhlo dle sekce 3.4 a jeho výsledky vidíme v tabulce 3.2.

Zde už se rozdíl mezi $>_{grlex}$ a $>_{grevlex}$ projevil více. Naše empirické ověření tedy opravdu ukazuje na to, že uspořádání $>_{grevlex}$ je z trojice $>_{lex}$, $>_{grlex}$, $>_{grevlex}$ pravděpodobně tím nejlepší kandidátem pro použití při řešení soustav polynomiálním rovnic. Volba $>_{grevlex}$ je zároveň v souladu s předchozími pracemi zabývajícími se algebraickou kryptoanalýzou za využití Gröbnerových bází (viz např. [7, 8]).

Proto i ve všech dalších měřeních v této kapitole budeme používat výhradně uspořádání $>_{grevlex}$.

3.6 Kombinační generátor – 18 bitů

V této sekci budeme pracovat s kombinačním generátorem složeným ze 4 LFSR. První dva LFSR mají po 5 bitech vnitřního stavu a pro jejich koeficienty zpětné vazby platí

$$(c_1, c_2, c_3, c_4, c_5) = (0, 1, 0, 0, 1).$$

Druhé dva LFSR mají po 4 bitech vnitřního stavu a jejich koeficienty zpětné vazby splňují

$$(c_1, c_2, c_3, c_4) = (1, 0, 0, 1).$$

Označme u_t^i t-tý výstupní bit i-tého LFSR. Pro t-tý výstupní bit generátoru platí

$$s_t = u_t^1 u_t^2 u_t^3 u_t^4.$$

Poznámka 3.6.1. Všimněme si, že funkce, která kombinuje výstupní bity jednotlivých LFSR,

$$f(u_t^1, u_t^2, u_t^3, u_t^4) = u_t^1 u_t^2 u_t^3 u_t^4,$$

není *vyvážená*. Pro 15 z 16 možných kombinací vstupů nabývá f hodnoty 0 a jen pro jedinou nabývá hodnoty 1. V keystreamu produkovaném generátorem tudíž budou výrazně převažovat nulové bity.

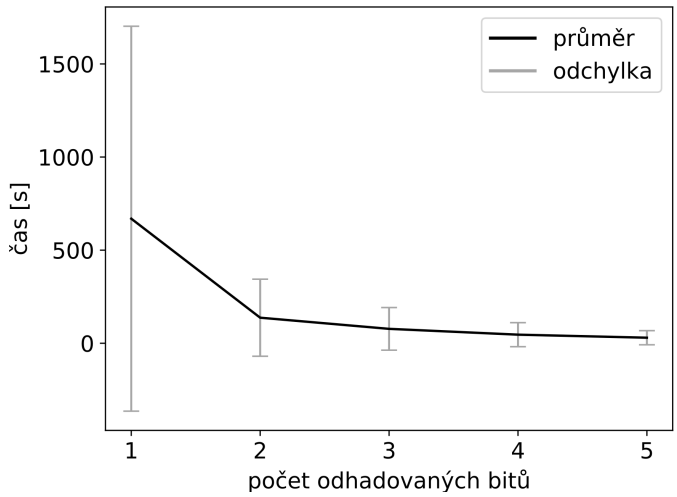
Funkce f je pro reálné použití krajně nevhodná, jelikož vyváženost je jedna ze základních vlastností, kterou od takové funkce požadujeme (viz sekci 2.1).

Nyní přejděme k jednotlivým variantám řešení, které jsme na tento generátor keystreamu použili.

3.6.1 Referenční výpočet

Zde jsme postupovali přesně tak, jak bylo popsáno v sekci 3.2.1. Výsledkem měření v souladu se sekcí 3.4 byl průměrný čas 1126.582 s a směrodatná odchylka 1707.533 s.

Naměřená odchylka ukazuje na velmi výrazné rozdíly v jednotlivých časech řešení. Bližší analýza výsledků ukazuje, že toto je způsobeno případy, kdy je vygenerovaný keystream složen pouze z nulových bitů. Řešení soustav polynomiálních rovnic, které odpovídají těmto případům, je výrazně časově náročnější, než je tomu u soustav ostatních.



Obrázek 3.1: Počáteční stav prvního LFSR – kombinační generátor, 18 bitů

3.6.2 Počáteční stav prvního LFSR

V rámci této varianty guess-and-determine za množinu „odhadovaných“ neznámých volíme prvních n bitů prvního LFSR generátoru keystreamu. Použijeme-li značení konzistentní s kapitolou 2, jedná se o bity $s_0, \ldots, s_{n-1}$ daného LFSR.

První LFSR má celkem 5 bitů vnitřního stavu, pracujeme tedy s $n \in \{1, 2, 3, 4, 5\}$.

Poznámka 3.6.2. Pro $n = 5$ máme jistotu, že v řešené soustavě polynomiálních rovnic jsou obsaženy pouze polynomy celkového stupně nejvýše 3. V obecném případě v ní mohou být polynomy celkového stupně až 4.

Závislost času řešení na hodnotě n při použití tohoto přístupu a dodržení postupu ze sekce 3.4 zachycuje graf 3.1.

3.6.3 Četnost proměnných

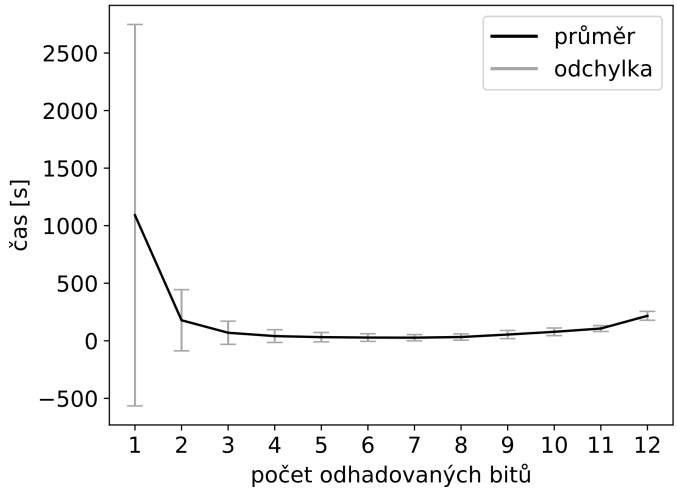
Tato varianta guess-and-determine staví na analýze soustavy polynomiálních rovnic, jež je řešena v rámci referenčního řešení. V této soustavě pro každou proměnnou (tj. pro každý bit počátečního stavu generátoru keystreamu) spočteme, v kolika monomech napříč všemi polynomy v soustavě se vyskytuje v nenulové mocnině.

Následně proměnné seřadíme od té nejčastější po nejméně častou. Jako množinu „odhadovaných“ neznámých v rámci guess-and-determine pak volíme prvních n z takto seřazených proměnných.

Konkrétně u tohoto kombinačního generátoru bychom teoreticky mohli volit libovolné $n \in \{1, \ldots, 18\}$. V našich měřeních jsme pracovali s $n \in \{1, \ldots, 12\}$.

pořadí	č. LFSR	č. bitu	četnost
1	3	2	181
2	4	2	181
3	3	0	174
4	4	0	174
5	3	1	155
6	4	1	155
7	1	3	145
8	2	3	145
9	1	1	134
10	2	1	134
11	3	3	132
12	4	3	132

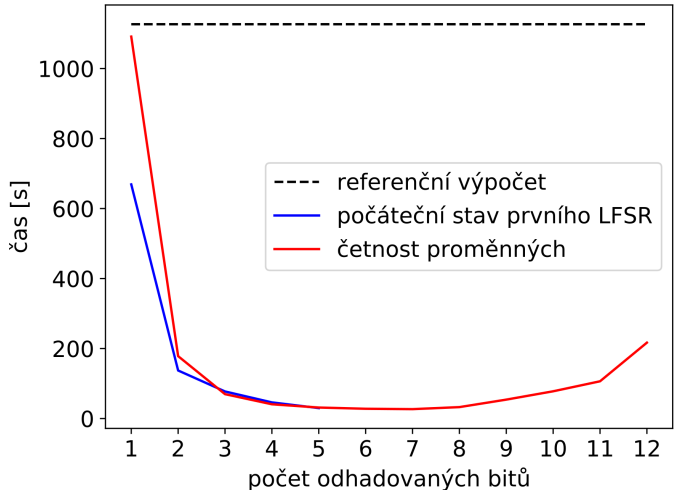
Tabulka 3.3: Četnost bitů počátečního stavu – kombinační generátor, 18 bitů



Obrázek 3.2: Četnost proměnných – kombinační generátor, 18 bitů

V tabulce 3.3 je k nalezení přehled 12 nejčastěji se vyskytujících bitů vnitřního stavu kombinačního generátoru. LFSR generátoru keystreamu číslujeme v rozmezí 1-4, jednotlivé bity každého LFSR číslujeme (konzistentně s definicí 2.0.1) od 0. Tj. v rozsahu 0-4 (pro první dva LFSR), respektive 0-3 (pro druhé dva LFSR).

Závislost času řešení na hodnotě n při použití tohoto přístupu a zachování postupu ze sekce 3.4 zachycuje graf 3.2.



Obrázek 3.3: Porovnání – kombinační generátor, 18 bitů

metoda	aritmetický průměr [s]	zrychlení	směrodatná odchylka [s]
referenční výpočet	1126,582	–	1707,533
počáteční stav 1. LFSR, $n = 5$	29,807	37,796	37,951
četnost proměnných, $n = 7$	26,833	41,985	27,261

Tabulka 3.4: Nejlepší výsledky – kombinační generátor, 18 bitů

3.6.4 Porovnání

Graf 3.3 umožňuje porovnat průběh průměrných časů metod založených na guess-and-determine s referenčním řešením.

V tabulce 3.4 jsou číselně zaznamenány nejlepší dosažené výsledky (ve smyslu nejnižšího průměrného času) všech metod použitých pro tento generátor keystreamu. Uvedené zrychlení je zrychlením dané metody vůči referenčnímu výpočtu, tj. poměr času referenčního výpočtu a času řešení danou metodou. Zopakujme, že popsaná měření se řídí sekcí 3.4.

3.7 Kombinační generátor – 64 bitů

Pro tuto sekci budeme uvažovat kombinační generátor složený ze 4 LFSR. Všechny LFSR mají po 16 bitech vnitřního stavu a jejich koeficienty zpětné vazby jsou dány vztahem

$$(c_1, \ldots, c_{16}) = (0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1).$$

Označme u_t^i t-tý výstupní bit i-tého LFSR. Pro t-tý výstupní bit kombinačního generátoru platí

$$s_t = f(u_t^1, u_t^2, u_t^3, u_t^4) = u_t^1 + u_t^2 u_t^3 + u_t^1 u_t^2 u_t^3 + u_t^1 u_t^3 u_t^4. \tag{3.2}$$

Poznámka 3.7.1. Funkce f, která kombinuje výstupní bity jednotlivých LFSR *je* vyvážená. Pro 8 z 16 možných kombinací vstupů nabývá hodnoty 0, pro 8 zbylých nabývá hodnoty 1.

Následují jednotlivé varianty řešení, které jsme pro tento kombinační generátor použili.

3.7.1 Referenční výpočet

Zde jsme postupovali v souladu se sekcemi 3.2.1 a 3.4. Výsledkem měření byl průměrný čas 817.260 s a směrodatná odchylka 50.134 s.

Poznámka 3.7.2. Všimněme si výrazného rozdílu směrodatných odchylek v porovnání se sekcí 3.6.1.

3.7.2 Počáteční stav třetího LFSR

Tato varianta metody guess-and-determine je analogií varianty popsané v sekci 3.6.2. Jediným rozdílem je volba LFSR, jehož bity počátečního stavu „hádáme“.

Myšlenka za volbou třetího LFSR staví na faktu, že výstupní bity tohoto LFSR vstupují do obou monomů celkového stupně 3 a navíc i do monomu celkového stupně 2 polynomu ve vztahu (3.2).

Délka třetího LFSR umožňuje volbu n (tj. počtu „odhadovaných“ bitů) v rozsahu $n \in \{1, \ldots, 16\}$. V provedených měřeních jsme pracovali s $n \in \{1, \ldots, 12\}$. Výsledek těchto měření respektive závislost času řešení na hodnotě n při dodržení postupu ze sekce 3.4 je popsána grafem 3.4.

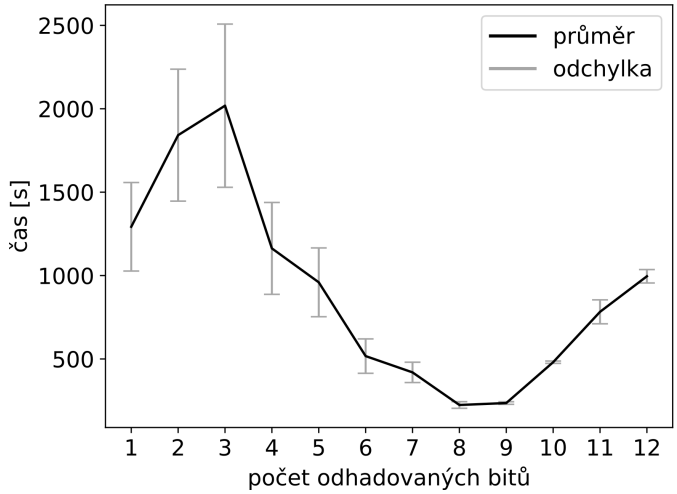
3.7.3 Četnost proměnných

Zde jsme postupovali zcela analogicky s variantou popsanou v sekci 3.6.3.

Bitová délka vnitřního stavu kombinačního generátoru nám dovoluje pracovat až s $n \in \{1, \ldots, 64\}$ proměnnými. V rámci měření jsme pracovali s $n \in \{1, \ldots, 12\}$.

Tabulka 3.5 obsahuje 12 nejčastěji se vyskytujících bitů vnitřního stavu kombinačního generátoru. LFSR, jimiž je kombinační generátor tvořen, číslujeme v rozsahu 1-4. Jednotlivé bity každého LFSR jsou číslovány v rozmezí 0-15.

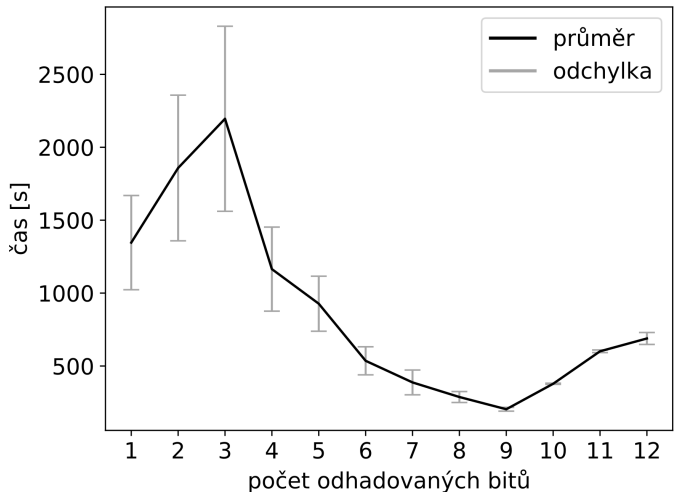
Závislost času řešení na hodnotě n při použití této varianty guess-and-determine a dodržení postupu ze sekce 3.4 je zachycena grafem 3.5.



Obrázek 3.4: Počáteční stav třetího LFSR – kombinační generátor, 64 bitů

pořadí	č. LFSR	č. bitu	četnost
1	3	5	5939
2	3	14	5926
3	3	0	5914
4	3	2	5808
5	3	1	5789
6	3	6	5780
7	1	5	5691
8	3	3	5690
9	1	14	5675
10	1	0	5663
11	3	15	5661
12	1	2	5562

Tabulka 3.5: Četnost bitů počátečního stavu – kombinační generátor, 64 bitů



Obrázek 3.5: Četnost proměnných – kombinační generátor, 64 bitů

3.7.4 Hodnota $u_t^1 u_t^3$

Připomeňme, že u_t^i značí t-tý výstupní bit i-tého LFSR. Množina neznámých, které „odhadujeme“ v rámci této varianty guess-and-determine, pak je množina součinů

$$\{u_t^1 u_t^3 \mid t \in T\},$$

kde $T \subseteq \{0, \ldots, 63\}$ je množina vybraných indexů bitů keystreamu.

Zafixujme množinu T a zvolme $t \in T$. Pak původní vztah (3.2) je zjednodušen jedním ze dvou způsobů. Za předpokladu $u_t^1 u_t^3 = 0$ platí

$$s_t = f_0(u_t^1, u_t^2, u_t^3, u_t^4) = u_t^1 + u_t^2 u_t^3.$$

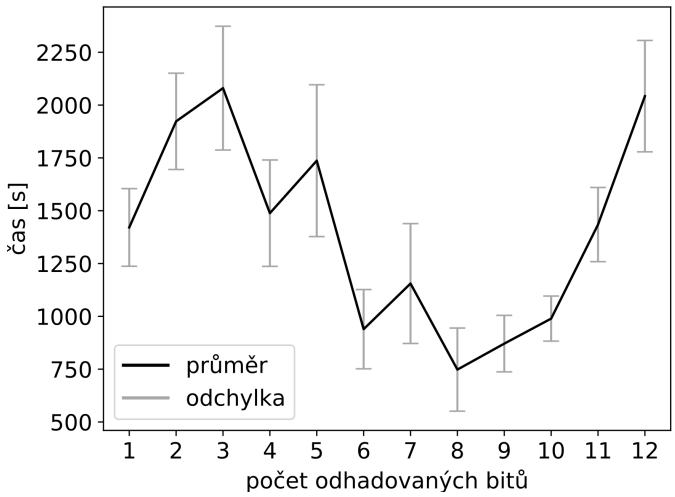
Oproti tomu pokud $u_t^1 u_t^3 = 1$, pak

$$s_t = f_1(u_t^1, u_t^2, u_t^3, u_t^4) = u_t^4 + 1.$$

Namísto původních polynomiálních rovnic s polynomy celkového stupně až 3 nyní v rovnicích odpovídajících vybraných bitům keystreamu (v podobě množiny T) dostáváme polynomy celkového stupně 2, nebo dokonce rovnice lineární.

Za každou takto zjednodušenou rovnici nám ovšem do soustavy navíc přibude jedna polynomiální rovnice s polynomem celkového stupně 2. Jedná se o samotný předpoklad

$$u_t^1 u_t^3 = 0, \quad \text{nebo} \quad u_t^1 u_t^3 = 1.$$



Obrázek 3.6: Hodnota $u_t^1 u_t^3$ – kombinační generátor, 64 bitů

3.7.4.1 Volba množiny T

V měřeních obsažených v této práci „nezahazujeme“ prvních m výstupních bitů generátoru keystreamu (v praxi se tak děje, viz sekci 2.1). V rovnicích odpovídajících prvním bitům keystreamu tudíž vystupuje jen relativně málo bitů vnitřního stavu generátoru. Další rovnice jsou v tomto smyslu postupně složitější a složitější.

Ve snaze o co nejvýraznější zjednodušení soustavy polynomiálních rovnic tudíž volíme rovnice postupně od poslední k první. Tedy volíme

$$T = \{63 - n + 1, \ldots, 63\},$$

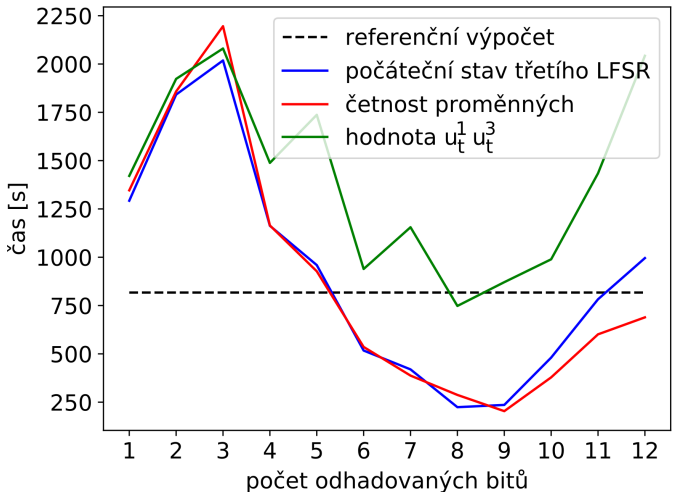
a to konkrétně pro $n \in \{1, \ldots, 12\}$.

Výsledek těchto měření provedených v souladu se sekcí 3.4 zachycuje graf 3.6.

3.7.5 Porovnání

Vzájemné srovnání průběhu průměrných časů metod založených na guess-and-determine s referenčním řešením je zachyceno v grafu 3.7.

V tabulce 3.6 jsou obsaženy nejlepší dosažené výsledky všech metod použitých pro tento kombinační generátor. Zopakujme, že popsaná měření se řídí sekcí 3.4.



Obrázek 3.7: Porovnání – kombinační generátor, 64 bitů

metoda	aritmetický průměr [s]	zrychlení	směrodatná odchylka [s]
referenční výpočet	817,260	–	50,134
počáteční stav 3. LFSR, $n = 8$	224,078	3,647	19,905
četnost proměnných, $n = 9$	203,492	4,016	12,977
hodnota $u_t^1 u_t^3$, $n = 8$	747,953	1,093	196,696

Tabulka 3.6: Nejlepší výsledky – kombinační generátor, 64 bitů

3.8 Filtr generátor – 72 bitů

V této sekci budeme pracovat s filtr generátorem o 72 bitech vnitřního stavu. Koeficienty zpětné vazby jeho (jediného) LFSR jsou dány vztahy

$$\begin{aligned}(c_1, \ldots, c_{18}) &= (0,0,0,1,0,0,0,1,0,0,0,1,0,0,1,0,0,1),\\ (c_{19}, \ldots, c_{36}) &= (0,0,1,1,1,0,1,0,1,0,1,1,1,1,0,0,1,0),\\ (c_{37}, \ldots, c_{54}) &= (0,1,0,0,0,0,0,1,0,0,1,1,0,0,0,0,0,0),\\ (c_{55}, \ldots, c_{72}) &= (0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1).\end{aligned}$$

Označme u_t t-tý bit vnitřního stavu filtr generátoru a s_t jeho t-tý výstupní bit (v souladu se sekcí 2.1.2). Potom platí

$$s_t = f(u_1, \ldots, u_{72}) = u_1 + u_{19}u_{37} + u_1u_{19}u_{37} + u_1u_{37}u_{55}. \tag{3.3}$$

Poznámka 3.8.1. Vztah (3.3) vychází ze vztahu (3.2). Pracujeme zde ale s odlišnými proměnnými. Ve funkci f ve vztahu (3.3) hraje roli vždy první bit

pořadí	č. bitu	četnost
1	45	1891
2	63	1876
3	62	1870
4	47	1869
5	41	1854
6	60	1846
7	46	1844
8	44	1812
9	48	1800
10	40	1793

Tabulka 3.7: Četnost bitů počátečního stavu – filtr generátor, 72 bitů

každé čtvrtiny vnitřního stavu filtr generátoru.

Namísto vztahu (3.3) bychom mohli psát

$$s_t = f(u_1, u_{19}, u_{37}, u_{55}) = u_1 + u_{19}u_{37} + u_1u_{19}u_{37} + u_1u_{37}u_{55}, \tag{3.4}$$

přičemž o takto zapsané funkci f můžeme říct, že *je* vyvážená.

Nyní přejděme k jednotlivým variantám řešení, které jsme na tento generátor keystreamu použili.

3.8.1 Referenční výpočet

Zde jsme postupovali tak, jak bylo popsáno v sekcích 3.2.1 a 3.4. Výsledkem byl průměrný čas 281.753 s a směrodatná odchylka 555.470 s.

3.8.2 Četnost proměnných

U této varianty guess-and-determine jsme postupovali naprosto analogicky s variantou popsanou v sekci 3.6.3.

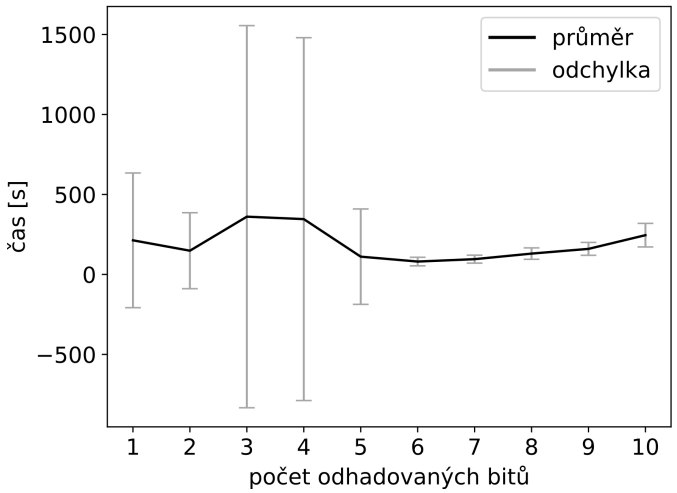
Bitová délka vnitřního stavu filtr generátoru nám umožňuje pracovat až s $n \in \{1, \ldots, 72\}$ proměnnými. V našich měřeních jsme pracovali v rozsahu $n \in \{1, \ldots, 10\}$.

V tabulce 3.7 je k nalezení přehled 10 nejčastěji se vyskytujících bitů vnitřního stavu filtr generátoru. Bity vnitřního stavu jsou číslovány v rozsahu 0-71.

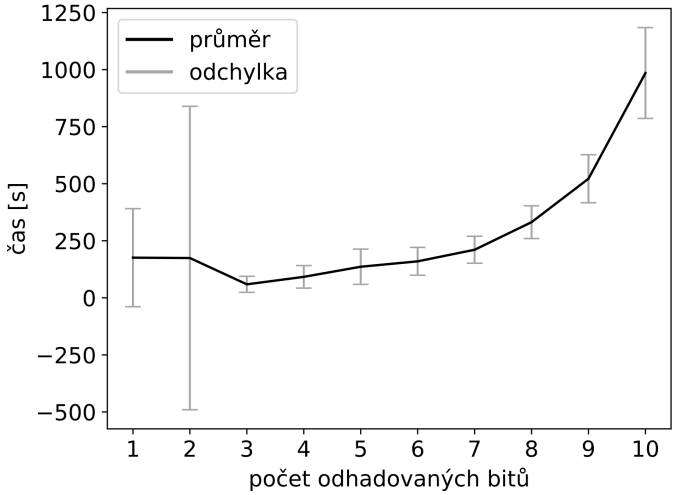
Závislost času řešení na hodnotě n při použití tohoto přístupu a dodržení postupu ze sekce 3.4 zachycuje graf 3.8.

3.8.3 Hodnota u_1u_{37}

Tato varianta je analogií varianty popsané v sekci 3.7.4. Jediným rozdílem je přejmenování proměnných stejným způsobem, jakým byly přejmenovány



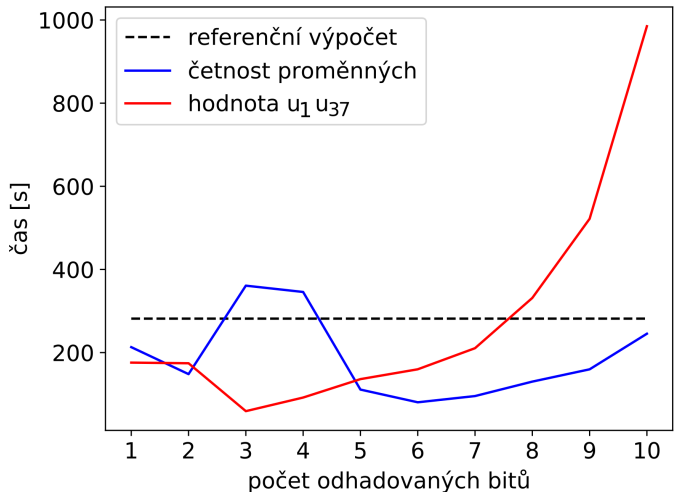
Obrázek 3.8: Četnost proměnných – filtr generátor, 72 bitů



Obrázek 3.9: Hodnota $u_1 u_{37}$ – filtr generátor, 72 bitů

mezi vztahy (3.2) a (3.4). Myšlenka tohoto přístupu i způsob volby množiny T zůstávají stejné.

Výsledky dosažené tímto přístupem a v souladu se sekcí 3.4 jsou zachyceny v grafu 3.9.



Obrázek 3.10: Porovnání – filtr generátor, 72 bitů

metoda	aritmetický průměr [s]	zrychlení	směrodatná odchylka [s]
referenční výpočet	281,753	–	555,470
četnost proměnných, $n = 6$	80,507	3,500	26,789
hodnota u_1u_{37}, $n = 3$	59,201	4,759	35,184

Tabulka 3.8: Nejlepší výsledky – filtr generátor, 72 bitů

3.8.4 Porovnání

Graf 3.10 srovnává průběh průměrných časů výpočtu metod založených na guess-and-determine s výsledkem referenčního výpočtu.

V tabulce 3.8 jsou číselně zaznamenány nejlepší dosažené výsledky jednotlivých metod použitých pro tento filtr generátor. Zopakujme, že popsaná měření se řídí sekcí 3.4.

Poznámka 3.8.2. Všimněme si rozdílu ve zrychlení, které přinesla metoda „odhadování“ hodnoty $u_t^1u_t^3$, respektive u_1u_{37} u kombinačního generátoru a u filtr generátoru. V případě kombinačního generátoru tato metoda jen těsně překonala referenční výpočet se zrychlením $1,093$, zatímco v případě filtr generátoru dosáhla zrychlení $4,759$ (viz tabulky 3.6 a 3.8).

Kapitola 4

Implementace

V rámci měření, která jsou v této práci obsažena, vzniklo několik skriptů a modulů. Popisu a vysvětlení těch nejdůležitějších je věnována tato kapitola.

4.1 Prerekvizity

Veškerý software použitý pro měření (viz kapitolu 3) byl vytvořen a vyzkoušen v prostředí zahrnujícím

- OS Ubuntu 20.04.4 LTS,
- Python 3.8.10,
- Magma V2.25-5.

4.1.1 Magma

Magma je (mimo jiné) algebraický software podporující symbolické výpočty. Z hlediska této práce je podstatné, že umožňuje pracovat s ideály a jejich varietami. Magma obsahuje implementaci algoritmů pro řešení soustav polynomiálních rovnic a konkrétně také algoritmu F4 pro výpočet Gröbnerovy báze polynomiálního ideálu (viz kapitolu 1). Kompletní dokumentace je k dispozici z [9].

4.2 Modul `magma_wrapper`

Měření byla postavena okolo jazyka Python. Skript napsaný v tomto jazyce byl vstupním místem, v němž bylo možné každé měření konfigurovat a spustit.

Modul `magma_wrapper` je, jak název napovídá, jakousi obálkou nad softwarem Magma, která umožňuje používat některé funkcionality softwaru Magma přímo z kódu jazyka Python.

4.2.1 Obecný princip

Celý postup, jímž tento modul software Magma zpřístupňuje, lze shrnout do několika kroků.

1. Je vytvořen skript čitelný softwarem Magma. Tento skript je uložen do souboru `magma.in`.
2. Software Magma je spuštěn jako podproces a na vstup je mu předán soubor `magma.in`.
3. Magma vykoná skript načtený ze souboru. Jeho součástí je vždy i to, aby byl výsledek zapsán do výstupního souboru `magma.out`.
4. Soubor `magma.out` je přečten, případně dál zpracován a výsledek je vrácen tomu, kdo modul `magma_wrapper` původně volal.

Modul `magma_wrapper` je souborem tříd, z nichž každá zprostředkovává jednu funkcionalitu softwaru Magma. Pro spuštění výpočtu je po vytvoření objektu příslušné třídy zapotřebí zavolat na něm bezparametrickou metodu `run`, která následně vrací požadované výsledky.

4.2.2 Parametr `namespace`

Výše popsaná forma „komunikace“ za využití souborů s sebou nese i určitá omezení. Předně je třeba ošetřit situaci, kdy je paralelně spuštěno vícero procesů využívajících modul `magma_wrapper`. V takovém případě by při pevně daných jménech souborů mohlo docházet k jejich nežádoucímu přepisování ostatními procesy.

A z tohoto důvodu je součástí každé funkce modulu `magma_wrapper` parametr `namespace`. Jeho hodnota je použita jako cesta, na níž jsou soubory `magma.in` a `magma.out` vytvořeny. V okamžiku, kdy je paralelně spuštěno vícero procesů volajících funkce modulu `magma_wrapper`, je tedy nutné, aby každý z nich využíval unikátní hodnotu parametru `namespace`.

Nyní přejděme k popisu samotných tříd, které jsou v modulu `magma_wrapper` obsaženy.

4.2.3 Třída `GetLfsrCoefficients`

LFSR je vhodné konstruovat tak, aby jeho polynom zpětné vazby byl primitivním polynomem (viz tvrzení 2.0.9). Třída `GetLfsrCoefficients` vrací koeficienty takového LFSR. Předpokládá se, že LFSR, jehož koeficienty chceme získat, je uvažováno nad $\mathbb{F}_2$.

4.2.3.1 Argumenty konstruktoru

- `length` – délka LFSR.
- `namespace` – viz sekci 4.2.2.

4.2.3.2 Výstup metody `run`

`tuple` o dvou prvcích

- `list` binárních koeficientů zpětné vazby LFSR (viz definici 2.0.1) v pořadí $c_L, c_{L-1}, \ldots, c_1$.
- `float` udávající čas výpočtu (v sekundách).

4.2.4 Třída `SimplifyPolynomial`

V rámci generování rovnic odpovídajících jednotlivým bitům keystreamu může být vhodné průběžně získané polynomy zjednodušovat. Třída `SimplifyPolynomial` toto umožňuje.

4.2.4.1 Argumenty konstruktoru

- `variables` – `list` řetězců označujících všechny proměnné, které se vyskytují v polynomu, jež chceme zjednodušit.
- `polynomial` – řetězec obsahující polynom, který chceme zjednodušit. Předpokladem je, že tento polynom je z $\mathbb{F}_2[x_1, \ldots, x_n]$.
- `namespace` – viz sekci 4.2.2.

4.2.4.2 Výstup metody `run`

`tuple` o dvou prvcích

- Řetězec obsahující zjednodušený polynom.
- `float` udávající čas výpočtu (v sekundách).

4.2.5 Třída `SolveEquations`

Třída `SolveEquations` zprostředkovává řešení polynomiálních rovnic nad $\mathbb{F}_2$. Při výpočtu se pracuje v monomiálním uspořádáním $>_{grevlex}$ (viz definici 1.2.10).

Z funkcí dostupných v softwaru Magma je zde využita mimo jiné funkce `Variety`, která provádí samotný výpočet řešení. Algoritmus, který je v rámci `Variety` zvolen, je závislý na podobě řešené soustavy. V případě, kdy by

všechny rovnice v soustavě byly nejvýše kvadratické, je použita optimalizovaná forma hledání hrubou silou. V ostatních případech je soustava vyřešena pomocí Gröbnerových bází. K výpočtu Gröbnerovy báze ideálu daného řešenou soustavou polynomiálních rovnic je použit algoritmus F4 [9].

4.2.5.1 Argumenty konstruktoru

- `equations` – `list` řetězců, z nichž každý obsahuje jeden polynom určující jednu rovnici.

  Předpokládá se, že všechny rovnice v soustavě jsou tvaru $f(x) = 0$. Chceme-li takovou rovnici zahrnout do soustavy, jako jeden z prvků tohoto argumentu předáme polynom $f(x)$.

- `variables` – `list` řetězců označujících proměnné, které se vyskytují v zadané soustavě.

- `namespace` – viz sekci 4.2.2.

4.2.5.2 Výstup metody `run`

`tuple` o dvou prvcích

- `list`, jehož každý prvek je `list` celých čísel (konkrétně 0 nebo 1 vzhledem k tomu, že soustava je řešena nad $\mathbb{F}_2$). Každý vnořený `list` odpovídá jednomu řešení zadané soustavy.

  Podoba řešení je dána argumentem `variables`. Každé řešení (vnořený `list`) obsahuje přesně tolik celých čísel, kolik proměnných bylo předáno na vstupu. Pořadí prvků řešení odpovídá pořadí proměnných v argumentu `variables`.

  V případě, že zadaná soustava nemá žádné řešení, vnější `list` je prázdný.

- `float` udávající čas výpočtu (v sekundách).

Příklad 4.2.1. Uvažujme soustavu

$$x + z = 0,$$
$$z + 1 = 0$$

jako ideál nad okruhem $\mathbb{F}_2[x, y, z]$. Řešení této soustavy s využitím `SolveEquations` je zachyceno v algoritmu 4.2.2.

Na výstupu pak dostaneme například `[[1, 0, 1], [1, 1, 1]], 0.08005402307026088`. Máme tedy dvě řešení

$$(x_0, y_0, z_0) = (1, 0, 1) \quad \text{a} \quad (x_1, y_1, z_1) = (1, 1, 1),$$

přičemž víme, že výpočet trval zhruba 0,08 vteřiny.

Algoritmus 4.2.2 Řešení soustavy polynomiálních rovnic

```
import magma_wrapper as mw

equations = ['x + z', 'z + 1']
variables = ['x', 'y', 'z']
namespace = 'example'

solver = mw.SolveEquations(equations, variables, namespace)
solutions, elapsed = solver.run()

print(f'{solutions}, {elapsed}')
```

4.3 Modul lfsr_keystream_generator

Teoreticky byl převod proudových šifer na soustavu rovnic popsán v sekci 2.2. Modul `lfsr_keystream_generator` tento proces implementuje v praxi.

V modulu jsou rozlišeny dva druhy generátorů keystreamu – filtr generátor a kombinační generátor (viz sekci 2.1). Oba tyto druhy jsou dále rozděleny na dvě podkategorie – binární a symbolický generátor.

4.3.1 Binární generátor

Binární generátor simuluje fungování reálného generátoru keystreamu. Má číselné (binární) koeficienty zpětné vazby i počáteční stav. A i jeho výstup je tedy binární posloupnost.

4.3.2 Symbolický generátor

Symbolický generátor má také číselné koeficienty zpětné vazby. Od binárního generátoru se liší v počátečním stavu. Ten je buď ryze symbolický (se všemi bity počátečního stavu se pracuje jako se symbolickými proměnnými), nebo zčásti symbolický a zčásti číselný (v případě, kdy část počátečního stavu známe, nebo ji hádáme). Výstupem symbolického generátoru pak je posloupnost polynomů obsahujících proměnné, které se vyskytovaly v počátečním stavu.

4.3.3 Využití pro měření

Způsob, kterým byly binární a symbolické generátory použity pro měření popsaná v kapitole 3, lze zapsat v několika krocích.

1. Jsou dány parametry generátoru (filtrační či kombinační funkce, koeficienty zpětné vazby a délka jednotlivých LFSR), jeho počáteční stav a množina známých bitů počátečního stavu.
2. Binární generátor za využití daných parametrů a počátečního stavu vygeneruje binární keystream potřebné délky.
3. Symbolický generátor za využití daných parametrů a množiny známých bitů počátečního stavu vygeneruje posloupnost polynomů odpovídajících jednotlivým bitům binárního keystreamu vygenerovaného v předchozím kroku.
4. Spojením polynomů získaných v kroku 3 s bity získanými v kroku 2 vznikne posloupnost polynomů reprezentující soustavu rovnic způsobem popsaným v sekci 4.2.5.1.
5. Soustava rovnic získaná v kroku 4 je vyřešena pomocí modulu `magma_wrapper` (viz sekci 4.2.5).

4.3.4 Závislost na modulu `magma_wrapper`

Složitost polynomů, které vznikají v rámci běhu symbolických generátorů, s počtem kroků generátoru rychle roste. Při naivním přístupu k práci s těmito polynomy mohou řetězce, jimiž jsou tyto polynomy zachyceny, narůst do neschůdné délky. Měření, která bychom se symbolickými generátory chtěli provádět, by pak byla limitována už samotným generování rovnic, ne jejich řešením.

Proto jsme v modulu `lfsr_keystream_generator` zvolili přístup průběžného zjednodušování vnitřního stavu. Ke zjednodušení dochází v každém kroku každého LFSR obsaženého v libovolném symbolickém generátoru keystreamu. Nová hodnota vstupující do LFSR je spočtena jako lineární kombinace prvků jeho vnitřního stavu (viz kapitolu 2). V případě LFSR se symbolickým vnitřním stavem je tato hodnota polynom. A tento polynom je před zapsáním do LFSR zjednodušen třídou `SimplifyPolynomial` (viz sekci 4.2.4).

Každý prvek počátečního stavu LFSR obsaženého v symbolickém generátoru keystreamu je buď číslo, nebo proměnná, a je tedy v nejjednodušším možném tvaru. A vzhledem k tomu, že i každý další prvek je před přidáním do vnitřního stavu zjednodušen, víme, že vnitřní stav LFSR obsaženého v symbolickém generátoru je nejjednodušší možný po každém vykonaném kroku.

Výše popsaný přístup s sebou samozřejmě nese určitou režii. Zejména v případě, kdy je zapotřebí jen několik málo kroků jednoduchého generátoru keystreamu, bude průběžné zjednodušování spíše na obtíž. Jakmile ale pracujeme s generátory řádově o desítkách bitů vnitřního stavu a pro každý bit vnitřního stavu generujeme rovnici, vede tento přístup k výraznému urychlení generování a zejména snížení objemu dat, která vygenerované rovnice zachycují.

4.3.5 Třída `BinaryFilterGenerator`

Tato třída reprezentuje binární filtr generátor.

4.3.5.1 Argumenty konstruktoru

- `coefficients` – `list` celých čísel reprezentujících koeficienty zpětné vazby. Koeficienty jsou zadány v pořadí $c_L, c_{L-1}, \ldots, c_1$ (viz definici 2.0.1).
- `initial_state` – `list` celých čísel (0 nebo 1) reprezentujících jednotlivé bity počátečního stavu. Bity jsou zadány v pořadí $s_0, s_1, \ldots, s_{L-1}$ (viz definici 2.0.1).
- `filter_function` – funkce f použitá pro výpočet výstupního bitu filtr generátoru na základě jeho aktuálního vnitřního stavu (viz sekci 2.1.2). Do funkce vstupují jako argumenty všechny bity vnitřního stavu.

4.3.5.2 Metoda `get_next`

Při volání této metody je vykonán jeden krok binárního generátoru. Tj. dojde ke změně vnitřního stavu a je vrácen jeden bit výstupní posloupnosti.

4.3.6 Třída `SymbolicFilterGenerator`

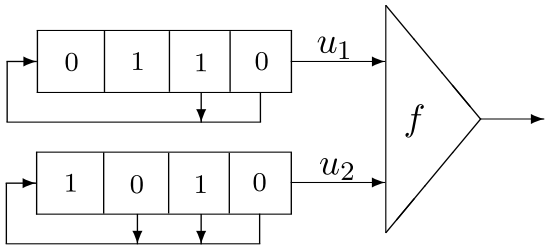
Tato třída reprezentuje symbolický filtr generátor.

4.3.6.1 Argumenty konstruktoru

- `coefficients` – viz stejnojmenný argument v sekci 4.3.5.1.
- `guessed_state` – `list` stejné délky jako argument `coefficients`, který obsahuje prvky 0, 1 nebo `None`. Číselné prvky značí, že příslušný bit je znám a nabývá zadané hodnoty. Hodnota `None` říká, že příslušný bit znám není a má se ve vygenerovaných rovnicích objevit jako proměnná.
- `filter_function` – viz stejnojmenný argument v sekci 4.3.5.1.
- `namespace` – argument potřebný pro použití modulu `magma_wrapper`. Viz sekci 4.2.2.

4.3.6.2 Metoda `get_next`

Zavoláním této metody je vykonán jeden krok symbolického generátoru. Dojde ke změně vnitřního stavu a je vrácen řetězec reprezentující další z posloupnosti polynomů.



Obrázek 4.1: Kombinační generátor z příkladu 4.3.1

4.3.6.3 Metoda `get_variables`

Tato metoda vrací `list` řetězců reprezentujících jednotlivé proměnné, s nimiž symbolický generátor pracuje.

4.3.7 Třída `BinaryCombinationGenerator`

Tato třída reprezentuje binární kombinační generátor.

4.3.7.1 Argumenty konstruktoru

- `lfsr_parameters` – `list`, jehož každý prvek je `tuple` o dvou prvcích. Každý tento `tuple` odpovídá jednomu LFSR obsaženému v kombinačním generátoru a význam jeho dvou prvků je
  1. stejný jako u argumentu `coefficients` v sekci 4.3.5.1,
  2. stejný jako u argumentu `initial_state` v sekci 4.3.5.1.
- `combination_function` – funkce f použitá pro výpočet výstupního bitu kombinačního generátoru na základě výstupních bitů dílčích LFSR (viz sekci 2.1.1). Funkce musí umět přijmout tolik argumentů, kolik je dílčích LFSR kombinačního generátoru.

4.3.7.2 Metoda `get_next`

Viz sekci 4.3.5.2.

Příklad 4.3.1. Uvažme kombinační generátor, který je ve svém počátečním stavu zachycen na obrázku 4.1. Pro funkci $f : \mathbb{F}_2^2 \to \mathbb{F}_2$ ať platí

$$f(u_1, u_2) = u_1 + u_2.$$

Simulace prvních 10 kroků tohoto generátoru s využitím `BinaryCombinationGenerator` je zachycena v algoritmu 4.3.2. Výstupem pak je posloupnost `[0, 0, 1, 1, 0, 0, 1, 1, 0, 1]`.

Algoritmus 4.3.2 Použití binárního kombinačního generátoru

```
import lfsr_keystream_generator as lkg

params = [
    ([1, 1, 0, 0], [0, 1, 1, 0]),
    ([1, 1, 1, 0], [0, 1, 0, 1])
]

def f(u1, u2):
    return u1 + u2

g = lkg.BinaryCombinationGenerator(params, f)
keystream = [g.get_next() for _ in range(10)]
print(keystream)
```

4.3.8 Třída `SymbolicCombinationGenerator`

Tato třída reprezentuje symbolický kombinační generátor.

4.3.8.1 Argumenty konstruktoru

- `symbolic_parameters` – `list`, jehož každý prvek je `tuple` o dvou prvcích. Každý tento `tuple` odpovídá jednomu LFSR obsaženému v kombinačním generátoru a význam jeho dvou prvků je
  1. stejný jako u argumentu `coefficients` v sekci 4.3.5.1,
  2. stejný jako u argumentu `guessed_state` v sekci 4.3.6.1.
- `combination_function` – viz stejnojmenný argument v sekci 4.3.7.1.
- `namespace` – viz stejnojmenný argument v sekci 4.3.6.1.

4.3.8.2 Metoda `get_next`

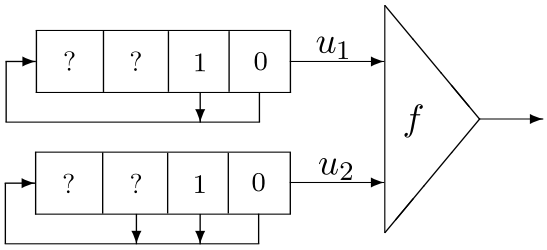
Viz sekci 4.3.6.2.

4.3.8.3 Metoda `get_variables`

Viz sekci 4.3.6.3.

Příklad 4.3.3. Uvažme kombinační generátor, který je ve svém počátečním stavu zachycen na obrázku 4.2. Známe tedy polovinu jeho vnitřního stavu. Pro funkci $f : \mathbb{F}_2^2 \to \mathbb{F}_2$ ať platí

$$f(u_1, u_2) = u_1 \cdot u_2.$$



Obrázek 4.2: Kombinační generátor z příkladu 4.3.3

Provedení prvních 4 kroků tohoto generátoru s využitím `SymbolicCombinationGenerator` je zachyceno v algroritmu 4.3.4. Výstupem je posloupnost `['0', '1', '(x2*x6)', '(x3*x7)']`.

Algoritmus 4.3.4 Použití symbolického kombinačního generátoru

```
import lfsr_keystream_generator as lkg

params = [
    ([1, 1, 0, 0], [0, 1, None, None]),
    ([1, 1, 1, 0], [0, 1, None, None])
]
namespace = 'namespace'

def f(u1, u2):
    return u1 * u2

g = lkg.SymbolicCombinationGenerator(params, f, namespace)
equations = [g.get_next() for _ in range(4)]
print(equations)
```

Kapitola 5

Související práce

Velmi podobný kontext jako tato práce nabízí [10]. Autoři pracují s binární proudovou šifrou založenou na LFSR a předpokládají útok typu known-plaintext. Nijak ovšem *nevyužívají* metodu guess-and-determine. Namísto ní se výpočet získané soustavy polynomiálních rovnic snaží urychlit rozšířením této soustavy o množství polynomiálních rovnic nízkého stupně.

Pro rovnici

$$f(u_t^1, \ldots, u_t^n) = s_t, \quad f \in \mathbb{F}_2[u_t^1, \ldots, u_t^n],$$

odpovídající t-tému bitu keystreamu kombinačního generátoru dojde k rozšíření soustavy o rovnici

$$f(u_t^1, \ldots, u_t^n) \cdot g(u_t^1, \ldots, u_t^n) = s_t \cdot g(u_t^1, \ldots, u_t^n),$$

kde $g \in \mathbb{F}_2[u_t^1, \ldots, u_t^n]$ je polynom splňující

- g je polynom nízkého celkového stupně,
- $f \cdot g$ je polynom celkového stupně výrazně nižšího, než je celkový stupeň polynomu f.

Takto rozšířená soustava polynomiálních rovnic je následně řešena pomocí Gröbnerových bází, *linearizace* nebo *algoritmu XL*. Metodika hledání výše popsaného polynomu g i výsledky dosažené touto metodou jsou v [10] také obsaženy.

5.1 Vektorový generátor keystreamu

Funkce, která kombinuje výstupní bity jednotlivých LFSR (v případě kombinačního generátoru) nebo jednotlivé bity vnitřního stavu (v případě filtr generátoru), nemusí být booleovská, tj.

$$f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2, \quad n \in \mathbb{N}.$$

Obecněji se může jednat o funkci

$$f : \mathbb{F}_2^n \longrightarrow \mathbb{F}_2^m, \quad 1 \leq m \leq n, \quad m, n \in \mathbb{N}. \tag{5.1}$$

Generátor keystreamu, jehož výstup je dán právě funkcí typu (5.1), je analyzován v [11]. Autor využívá metodu guess-and-determine, „odhaduje“ některé bity vnitřního (ne nutně počátečního) stavu generátoru a s jejich pomocí tvoří soustavu lineárních rovnic. Jejím řešením jsou bity počátečního stavu generátoru.

Tento přístup lze teoreticky použít i pro $m = 1$. Jeho efektivita tím ale klesne.

5.2 Sfinks

Proudovou šifru *Sfinks* lze popsat jako filtr generátor o 256 bitech vnitřního stavu s koeficienty zpětné vazby takovými, že perioda podkladového LFSR je maximální možná. Pro kryptoanalýzu této šifry je v [12] využito několika přístupů z [10], z nichž jeden je popsán v úvodu této kapitoly. Analýza v [12] ukazuje *Sfinks* jako šifru nedostatečně bezpečnou.

5.3 A5/2

V [13] se autoři věnují algebraické kryptoanalýze proudové šifry *A5/2* za využití metody guess-and-determine. V textu vysvětlují volbu bitů vnitřního stavu šifry pro „odhadnutí“. Na tomto útoku pak dále staví [14]. Zde jsou pro řešení získané soustavy polynomiálních rovnic namísto *linearizace* použity Gröbnerovy báze, jejichž výpočet je proveden algoritmem F4 (1.6.1).

5.4 NFSR

Algebraická kryptoanalýza proudové šifry využívající posuvný registr s *nelineární* zpětnou vazbou (NFSR) je obsahem [15]. Autoři se zaměřují konkrétně na rodinu proudových šifer *Grain*. Výsledkem jejich analýzy je počet bitů vnitřního stavu šifry, který je zapotřebí „odhadnout“ v rámci metody guess-and-determine tak, aby na stroji s danými parametry bylo možné dopočítat získanou soustavu polynomiálních rovnic.

Zároveň představují přístup, jakým se lze efektivněji řešit soustavu polynomiálních rovnic, která obsahuje polynomy o vysokém i nízkém celkovém stupni.

5.5 DES

V [16] se autoři zaobírají algebraickou kryptoanalýzou *blokové* šifry *DES*. Ačkoli popsaný převod šifry na soustavu polynomiálních rovnic není v kontextu

naší práce příliš dobře využitelný, je zde k nalezení srovnání několika způsobů řešení získané soustavy. Konkrétně jsou to Gröbnerovy báze, eliminační metody a řešiče *SAT*.

Závěr

V teoretické části práce jsme nejprve zavedli algebraické pojmy vedoucí k definici Gröbnerovy báze polynomiálního ideálu. Následně jsme se zaměřili na algoritmus F4. Podrobně jsme popsali algoritmus samotný, všechny jeho dílčí části i způsob, jakým ho lze použít pro řešení soustavy polynomiálních rovnic.

Dále jsme se věnovali konstrukci proudových šifer za využití LFSR. Jednalo se o dva druhy proudových šifer – kombinační generátor a filtr generátor. Zároveň jsme předvedli, jak popsat keystream vygenerovaný takto získanou proudovou šifrou pomocí soustavy polynomiálních rovnic.

Praktická část práce sestávala z experimentálního vyhodnocení rychlosti výpočtu několika variant metody guess-and-determine. Pracovali jsme se dvěma kombinačními generátory a jedním filtr generátorem. Konkrétně filtr generátor měl 72 bitů vnitřního stavu a byl v tomto smyslu největším generátorem keystreamu, jež se nám podařilo prolomit. Pro každý generátor keystreamu jsme sestavili sadu odpovídajících soustav polynomiálních rovnic. Na nich jsme porovnali metody založené na guess-and-determine se základním řešením využívajícím čistě algoritmus F4. U všech těchto generátorů keystreamu jsme na vytvořené sadě soustav polynomiálních rovnic oproti základnímu řešení dosáhli alespoň čtyřnásobného zrychlení a u jednoho z nich jsme překonali dokonce zrychlení čtyřicetinásobné.

V rámci praktické části práce jsme také vytvořili dva moduly v programovacím jazyce Python. První z těchto modulů slouží ke generování soustav polynomiálních rovnic na základě parametrů proudové šifry založené na LFSR. Druhý zprostředkovává některé funkcionality softwaru Magma. Zejména pak nástroje pro řešení soustav polynomiálních rovnic.

Literatura

[1] Canteaut, A.: Stream Cipher. 01 2011, doi:10.1007/0-387-23483-7_412.

[2] Cox, D. A.; Little, J.; O'Shea, D.: *Ideals, Varieties, and Algorithms*. Springer International Publishing, 2015, doi:10.1007/978-3-319-16721-3. Dostupné z: https://doi.org/10.1007/978-3-319-16721-3

[3] Buchberger, B.: Gröbner Bases and Systems Theory. *Multidimensional Systems and Signal Processing*, ročník 12, 07 2001: s. 223–251, doi:10.1023/A:1011949421611.

[4] Clark, P. L.: Reduced Row Echelon Form. 2013. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.378.2325&rep=rep1&type=pdf

[5] Nair, M. T.; Singh, A.: *Linear algebra*. Springer Singapore, 2018, doi:10.1007/978-981-13-0926-7. Dostupné z: https://doi.org/10.1007/978-981-13-0926-7

[6] Kaliski, B.: *Primitive Element*. Boston, MA: Springer US, 2011, ISBN 978-1-4419-5906-5, s. 965–965, doi:10.1007/978-1-4419-5906-5_426. Dostupné z: https://doi.org/10.1007/978-1-4419-5906-5_426

[7] Faugère, J.-C.; Spaenlehauer, P.-J.: Algebraic Cryptanalysis of the PKC'2009 Algebraic Surface Cryptosystem. In *Public Key Cryptography – PKC 2010*, editace P. Q. Nguyen; D. Pointcheval, Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, ISBN 978-3-642-13013-7, s. 35–52, doi:10.1007/978-3-642-13013-7_3.

[8] Bardet, M.; Briaud, P.; Bros, M.; aj.: An Algebraic Attack on Rank Metric Code-Based Cryptosystems. In *Advances in Cryptology – EUROCRYPT 2020*, editace A. Canteaut; Y. Ishai, Cham: Springer International Publishing, 2020, ISBN 978-3-030-45727-3, s. 64–93, doi:10.1007/978-3-030-45727-3_3.

[9] Computational Algebra Group, School of Mathematics and Statistics, University of Sydney: Magma. (navštíveno: 01.03.2022). Dostupné z: `http://magma.maths.usyd.edu.au/magma/documentation/`

[10] Courtois, N. T.; Meier, W.: Algebraic Attacks on Stream Ciphers with Linear Feedback. In *Advances in Cryptology — EUROCRYPT 2003*, editace E. Biham, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ISBN 978-3-540-39200-2, s. 345–359, doi:10.1007/3-540-39200-9_21.

[11] Pasalic, E.: On Guess and Determine Cryptanalysis of LFSR-Based Stream Ciphers. *IEEE Transactions on Information Theory*, ročník 55, č. 7, 2009: s. 3398–3406, doi:10.1109/TIT.2009.2021316.

[12] Courtois, N. T.: Cryptanalysis of Sfinks. In *Information Security and Cryptology - ICISC 2005*, editace D. H. Won; S. Kim, Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, ISBN 978-3-540-33355-5, s. 261–269, doi:10.1007/11734727_22.

[13] Barkan, E.; Biham, E.; Keller, N.: Instant Ciphertext-Only Cryptanalysis of GSM Encrypted Communication. In *Advances in Cryptology - CRYPTO 2003*, editace D. Boneh, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, ISBN 978-3-540-45146-4, s. 600–616, doi:10.1007/978-3-540-45146-4_35.

[14] Afzal, M.; Masood, A.; Shehzad, N.: Improved Results on Algebraic Cryptanalysis of A5/2. In *Global E-Security*, editace H. Jahankhani; K. Revett; D. Palmer-Brown, Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, ISBN 978-3-540-69403-8, s. 182–189, doi:10.1007/978-3-540-69403-8_22.

[15] Afzal, M.; Masood, A.: Algebraic Cryptanalysis of A NLFSR Based Stream Cipher. In *2008 3rd International Conference on Information and Communication Technologies: From Theory to Applications*, 2008, s. 1–6, doi:10.1109/ICTTA.2008.4530286.

[16] Courtois, N. T.; Bard, G. V.: Algebraic Cryptanalysis of the Data Encryption Standard. In *Cryptography and Coding*, editace S. D. Galbraith, Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, ISBN 978-3-540-77272-9, s. 152–169, doi:10.1007/978-3-540-77272-9_10.

Příloha A

Obsah přiloženého CD

- `src` ..... zdrojové kódy
  - `measurement` ..... zdrojové kódy měření popsaných v kapitole 3
    - `lfsr_keystream_generator.py` ..... modul popsaný v sekci 4.3
    - `lfsr_keystream_generator_tests.py` ..... sada testů k modulu
    - `magma_wrapper.py` ..... modul popsaný v sekci 4.2
    - `measurement_cg.py` ..... viz sekci A.1
    - `measurement_cg_guess_function.py` ..... viz sekci A.2
    - `measurement_fg.py` ..... viz sekci A.3
    - `measurement_fg_guess_function.py` ..... viz sekci A.4
    - `measurement_tools.py` ..... pomocný modul
  - `thesis` ..... zdrojové kódy práce ve formátu LaTeX
- `text` ..... text práce
  - `DP_Soukup_Jiri_2022.pdf` ..... text práce ve formátu PDF

A.1 `measurement_cg.py`

Tento skript spouští na kombinačním generátoru měření buďto referenčního výpočtu, nebo varianty metody guess-and-determine, která využítá „hádání“ bitů vnitřního stavu generátoru. Jedná se tedy o měření popsaná v sekcích 3.6.1, 3.6.2, 3.6.3, 3.7.1, 3.7.2 a 3.7.3.

A.2 `measurement_cg_guess_function.py`

Tento skript spouští na kombinačním generátoru měření varianty guess-and-determine, která využívá „hádání“ hodnoty vyskytující se ve funkci kombinující výstupní bity jednotlivých LFSR. Tj. měření popsané v sekci 3.7.4.

A.3 measurement_fg.py

Tento skript je analogií skriptu `measurement_cg.py` (viz sekci A.1) s tím rozdílem, že namísto kombinačního generátoru pracuje s filtr generátorem. Byl použit pro měření zachycená v sekcích 3.8.1 a 3.8.2.

A.4 measurement_fg_guess_function.py

Tento skript je analogií skriptu `measurement_cg_guess_function.py` (viz sekci A.2), jen namísto kombinačního generátoru pracuje s filtr generátorem. Byl použit pro měření popsané v sekci 3.8.3.