FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ČVUT V PRAZE

Zadání diplomové práce

Název:	Redukování předefinovaných systémů polynomiálních rovnic odvozených ze zjednodušených variant AES
Student:	Bc. Jana Berušková
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

Algebraická kryptoanalýza založena na Groebnerových bázích je moderní a efektivní přístup používaný k nalezení tajného klíče. Algebraická kryptoanalýza je obvykle založena na útocích se znalosti otevřeného textu. Pokud je k dispozici více párů otevřených textů a odpovídajících šifrovacích textů, lze odvodit více polynomiálních rovnic. Hlavním cílem této práce je navrhnout a implementovat metodu redukce a zjednodušení tohoto předefinovaného systému polynomiálních rovnic a ve výsledku tak urychlit výpočet Groebnerových bází.

Podrobné pokyny:
1) Popište Groebnerovy báze a algoritmy pro jejich výpočet.
2) Převeďte zjednodušenou variantu AES na systém polynomiálních rovnic.
3) Navrhněte a implementujte metodu redukce a zjednodušení předefinovaného systému.
4) Použijte vhodný program (např. Magma) k výpočtu Groebnerových bází a proveďte diskusi o výsledcích.

Elektronicky schválil/a prof. Ing. Róbert Lórencz, CSc. dne 16. dubna 2021 v Praze.

FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ
ČVUT V PRAZE

Diplomová práce

Redukování předefinovaných systémů polynomiálních rovnic odvozených ze zjednodušených variant AES

Bc. Jana Berušková

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

14. září 2022

Poděkování

Ráda bych touto cestou poděkovala především svému vedoucímu Mgr. Martinu Jurečkovi, Ph.D. za jeho cenné rady a trpělivost a také Ing. Markovi Bielikovi za jeho předešlou práci a pomoc při jejím rozboru.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů 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 předpisů. V souladu s ust. § 2373 odst. 2 zákona č. 89/2012 Sb., občanský zákoník, ve znění pozdějších předpisů, tímto uděluji nevýhradní oprávnění (licenci) k užití této mojí práce, a to včetně všech počítačových programů, jež jsou její součástí či přílohou a veškeré jejich dokumentace (dále souhrnně jen „Dílo“), a to všem osobám, které si přejí Dílo užít. Tyto osoby jsou oprávněny Dílo užít jakýmkoli způsobem, který nesnižuje hodnotu Díla, avšak pouze k nevýdělečným účelům. Toto oprávnění je časově, teritoriálně i množstevně neomezené.

V Praze dne 14. září 2022 . . . . . .

Odkaz na tuto práci

Berušková, Jana. *Redukování předefinovaných systémů polynomiálních rovnic odvozených ze zjednodušených variant AES*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2022. Dostupné z: ⟨https://github.com/berusjan/RedPolSysAES⟩.

Abstrakt

Tato práce se zabývá možnostmi zefektivnění výpočtu tajného klíče šifry AES (či jejích zjednodušených verzí) pomocí různých metod redukce předefinovaného polynomiálního systému rovnic, jež tuto šifru modelují. Redukci provedeme pomocí výběru a vzájemného sčítání takových polynomů, aby se zjednodušil celkový výpočet. Jednou z možností je například sčítání polynomů se stejným nejvýznamnějším monomem nebo nalezení co nejpodobnějších polynomů pomocí klastrování.

Klíčová slova AES, algebraická kryptoanalýza, Gröbnerovy báze, klastrování, PAM, LSH, vedoucí monom

Abstract

This master thesis deals with the possibility of increasing efficiency of the calculation of the secret key of the AES cipher (or its small scale derivatives) by different types of reduction of overdefined system of polynomial equations, which models the cipher. The reduction is done by selecting and xoring such polynomials, to simplify the overall calculation. One possibility is, for example, xoring polynomials with the same most-significant monomial or finding the most similar polynomials using clustering.

Keywords AES, algebraic cryptoanalysis, Gröbner bases, clustering, PAM, LSH, leading monom

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Žijeme v době, kdy důležitost kryptografie stále stoupá. Dříve lidé stavěli vysoké zdi, aby ochránili svůj majetek, dnes jsou však jednou z nejcennějších komodit data a informace. Právě proto je třeba je i náležitě chránit. V současnosti je v oblasti kybernetické bezpečnosti jedna z nejpoužívanějších šifer šifra AES.

Abychom měli jistotu, že tato šifra stále plní svou funkci, musíme ji neustále podrobovat zkouškám odolnosti. K tomu velice přispívá obor algebraické kryptoanalýzy. Ten se zabývá snahou prolomit danou šifru pomocí jejích algebraických modelů, které lze matematicky vyřešit. Otázkou bezpečnosti však zůstává, za jak dlouho jsme schopní toto řešení nalézt.

My se v této práci zabýváme právě problematikou rychlosti výpočtu tajného klíče pomocí výpočtu polynomiálního systému rovnic popsaného Ing. Bielikem. V 1. kapitole si představíme odborné termíny, nezbytné pro pochopení matematického základu potřebného k naší práci. Popíšeme si Gröbnerovy báze a algoritmus F4, který se používá pro jejich výpočet.

Ve 2. kapitole se pak budeme zabývat šifrou AES. Nejprve popíšeme celý algoritmus šifrování a představíme si její zjednodušené varianty, se kterými budeme provádět naše experimenty. Poté si popíšeme systém polynomiálních rovnic, kterými se tyto varianty dají namodelovat, a skripty pana Bielika [1], které tento systém umí vytvořit a spočítat.

Ve 3. kapitole se pak zaměříme na možnosti zefektivnění tohoto výpočtu pomocí předefinování daného systému a jeho následnou redukcí. Představíme si 3 možnosti výběru takových polynomů, jejichž vzájemný součet (neboli xor) vytvoří polynomy vhodnější pro výpočet Gröbnerovy báze pomocí algoritmu F4 a tím urychlí nalezení tajného klíče.

V poslední kapitole pak popíšeme všechny provedené experimenty a povedeme diskuzi o výsledcích.

Kapitola 1

Úvod do používané matematiky

V této kapitole uvádíme základní vymezení a definování všech matematických pojmů, které budou v této práci použity. Zaměříme se především na oblast Gröbnerových bází a jejich výpočet pomocí algoritmu F4.

Většina zde vypsaných definic je převzata z [2].

1.1 Polynomy, ideály, variety

Nejprve je potřeba si zadefinovat, v jakém matematickém prostoru se budeme po celou dobu pohybovat.

Definice 1.1.1. Pojmem **monom** označíme součin ve tvaru

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

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

Definice 1.1.2. Polynom f v proměnných $x_1, ..., x_n$ nad tělesem k je konečná lineární kombinace monomů s koeficienty v k. Polynom f zapíšeme jako

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

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

Takovouto množinu $k[x_1, ..., x_n]$ pak nazýváme **polynomiální okruh**. Přesnou definici lze najít v [2, příloha A].

Definice 1.1.3. Nechť $f = \sum_\alpha a_\alpha x^\alpha$ je polynom z $k[x_1, ..., x_n]$.

(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** polynomu $f \neq 0$, který označujeme jako $\deg(f)$, je největší $|\alpha|$ takové, že koeficient a_α je nenulový. Celkový stupeň nulového polynomu není definován.

Definice 1.1.4. Nechť $f_1, ..., f_s$ jsou polynomy z $k[x_1, ..., x_n]$. Pak položíme

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

$V(f_1, ..., f_s)$ nazveme **afinní varietou** danou $f_1, ..., f_s$.

Varieta $V(f_1, ..., f_s) \subseteq k^n$ je množina všech řešení soustavy polynomiálních rovnic $f_1(x_1, ..., x_n) = ... = f_s(x_1, ..., x_n) = 0$.

Lemma 1.1.5. *Pokud $V, W \subseteq k^n$ jsou afinní variety, pak i $V \cup W$ a $V \cap W$ jsou afinní variety.*

Důkaz je popsán v [2, str. 11].

Definice 1.1.6. Mějme podmnožinu $I \subseteq k[x_1, ..., x_n]$. I je **ideál**, pokud

(i) $0 \in I$,

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

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

Definice 1.1.7. Nechť $f_1, ..., f_s$ jsou polynomy z $k[x_1, ..., x_n]$. Pak položíme

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

Lemma 1.1.8. *Pokud $f_1, ..., f_s \in k[x_1, ..., x_n]$, pak $\langle f_1, ..., f_s \rangle$ je ideálem $k[x_1, ..., x_n]$. Tento ideál nazveme **ideál generovaný** $f_1, ..., f_s$.*

Tvrzení 1.1.9. *Pokud $f_1, ..., f_s$ a $g_1, ..., g_t$ jsou báze stejného ideálu $k[x_1, ..., x_n]$, takže $\langle f_1, ..., f_s \rangle = \langle g_1, ..., g_t \rangle$, pak i $\mathbf{V}(f_1, ..., f_s) = \mathbf{V}(g_1, ..., g_t)$.*

Důkaz vyplývá z definic variety a ideálu a není proto třeba jej zde uvádět.

Definice 1.1.10. Nechť $V \subseteq k^n$ je afinní varieta. Potom položíme

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

Lemma 1.1.11. *Pokud $V \subseteq k^n$ je afinní varieta, pak $\mathbf{I}(V) \subseteq k[x_1, ..., x_n]$ je ideál. Ideál $\mathbf{I}(V)$ nazveme* ***ideál variety*** *V .*

Důkaz je popsán v [2, str. 32].

Definice 1.1.12. Ideál $I \subseteq k[x_1, ..., x_n]$ je **monomiální ideál**, pokud existuje množina $A \subseteq \mathbb{Z}^n_{\geq 0}$ taková, že I se skládá ze všech polynomů, které jsou konečnou sumou tvaru $\sum_{\alpha \in A} h_\alpha x^\alpha$, kde $h_\alpha \in k[x_1, ..., x_n]$. V tomto případě píšeme, že $I = \langle x^\alpha \mid \alpha \in A \rangle$.

1.2 Monomiální uspořádání

Abychom mohli zavést pojem monomiální uspořádání, musíme nejprve vysvětlit, jak se chová lineární uspořádání obecně. Relace $>$ na $\mathbb{Z}^n_{\geq 0}$ je lineární uspořádání, pokud pro každou dvojici $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$ platí právě jedno ze tří tvrzení: $\alpha > \beta$ nebo $\alpha = \beta$ nebo $\alpha < \beta$.

Definice 1.2.1. Monomiální uspořádání $>$ na $k[x_1, ..., x_n]$ je relace na $\mathbb{Z}^n_{\geq 0}$ nebo také na množině monomů $x^\alpha, \alpha \in \mathbb{Z}^n_{\geq 0}$, která splňuje:

(i) $>$ je lineární uspořádání na $\mathbb{Z}^n_{\geq 0}$.

(ii) Pokud $\alpha > \beta$ a $\gamma \in \mathbb{Z}^n_{\geq 0}$, pak $\alpha + \gamma > \beta + \gamma$.

(iii) $>$ je dobré uspořádání na $\mathbb{Z}^n_{\geq 0}$. To znamená, že každá neprázdná množina $\mathbb{Z}^n_{\geq 0}$ má nejmenší prvek vzhledem k $>$. Tj. pro každou neprázdnou množinu $A \subseteq \mathbb{Z}^n_{\geq 0}$ existuje $\alpha \in A$ takové, že $\beta > \alpha$ pro každé $\beta \in A, \beta \neq \alpha$.

Tímto jsme si zavedli obecný pojem monomiálního uspořádání a nyní si uvedeme několik příkladů takových uspořádání, které budeme v práci požívat.

Definice 1.2.2. Lexikografické uspořádání. Nechť $\alpha = (\alpha_1, ..., \alpha_n)$ a $\beta = (\beta_1, ..., \beta_n)$ jsou z $\mathbb{Z}^n_{\geq 0}$. Řekneme, že $\alpha >_{lex} \beta$, pokud první nenulový prvek zleva rozdílu $\alpha - \beta \in \mathbb{Z}^n$ je kladný. Budeme psát $x^\alpha >_{lex} x^\beta$, pokud $\alpha >_{lex} \beta$.

Uvedeme několik příkladů:
a. $(3,2,4) >_{lex} (1,7,4)$ protože $\alpha - \beta = (2,-5,0)$.
b. $(6,5,7) >_{lex} (6,3,9)$ protože $\alpha - \beta = (0,2,-2)$.
c. $(1,4,3) <_{lex} (8,5,2)$ protože $\alpha - \beta = (-7,-1,1)$.

Definice 1.2.3. Odstupňované lexikografické uspořádání. Nechť $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$. Řekneme, že $\alpha >_{grlex} \beta$, pokud

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

nebo

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

Zde vidíme, že odstupňované lexikografické uspořádání řadí nejprve podle celkového stupně monomu. Pokud je stejný, tak až poté podle lexikografického uspořádání.

A opět uvedeme dva příklady:
a. $(5,3,8) >_{grlex} (7,4,1)$ protože $16 > 12$.
b. $(5,3,2) >_{grlex} (2,7,1)$ protože $10 = 10$ a $(5,3,2) >_{lex} (2,7,1)$.

Definice 1.2.4. Obrácené odstupňované lexikografické uspořádání. Nechť $\alpha, \beta \in \mathbb{Z}^n_{\geq 0}$. Řekneme, že $\alpha >_{grevlex} \beta$, pokud

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

nebo

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

Podobně jako odstupňované lexikografické uspořádání i to obrácené řadí nejprve podle celkového stupně monomu a když ten je stejný, tak poté již nastává rozdíl.

Pro obě varianty opět uvedeme příklad:
a. $(5,3,8) >_{grevlex} (7,4,1)$ protože $16 > 12$.
b. $(5,3,2) >_{grevlex} (2,3,5)$ protože $10 = 10$ a $(5,3,2) - (2,3,5) = (3,0-3,)$.

Definice 1.2.5. Nechť $f = \sum_\alpha a_\alpha x^\alpha$ je nenulový polynom z $k[x_1, ..., x_n]$ a ať $>$ je monomiální uspořádání.

(i) **Multistupeň** polynomu f je

$$\text{multideg}(f) = \max(\{\alpha \in \mathbb{Z}^n_{\geq 0} \mid a_\alpha \neq 0\}),$$

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

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

Pro ukázku uvedeme celkový příklad. Mějme polynom $f = 6x^7y^9z^4 + x^2y^5z^3$ a uvažujme lexikografické uspořádání. Z toho vypočítáme:
$\text{multideg}(f) = (7, 9, 4)$
$\text{LC}(f) = 6$
$\text{LM}(f) = x^7y^9z^4$
$\text{LT}(f) = 6x^7y^9z^4$

Lemma 1.2.6. *Nechť $f, g \in k[x_1, ..., x_n]$ jsou nenulové polynomy. Pak:*

(i) $\text{multideg}(fg) = \text{multideg}(f) + \text{multideg}(g)$.

(ii) Pokud $f+g \neq 0$, pak $\text{multideg}(f+g) \leq \max(\text{multideg}(f), \text{multideg}(g))$. *Pokud je navíc* $\text{multideg}(f) \neq \text{multideg}(g)$, *pak* $\text{multideg}(f + g) = \max(\text{multideg}(f), \text{multideg}(g))$.

1.3 Algoritmus dělení polynomů

Věta 1.3.1. ***Dělení v*** $k[x_1, ..., x_n]$**.** *Nechť* $>$ *je monomiální uspořádání na* $\mathbb{Z}^n_{\geq 0}$ *a nechť* $F = (f_1, ..., f_s)$ *je usporádaná s-tice polynomů z* $k[x_1, ..., x_n]$*. Pak každý polynom* $f \in k[x_1, ..., x_n]$ *můžeme zapsat ve tvaru*

$$f = q_1 f_1 + ... + q_s f_s + r,$$

kde $q_i, r \in k[x_1, ..., x_n]$ *a zároveň buď* $r = 0$ *nebo* r *je lineární kombinací (s koeficienty v* k*) monomů, z nichž žádný není dělitelný žádným termem z* $\mathrm{LT}(f_1), ..., \mathrm{LT}(f_s)$*. Hodnotu* r *nazveme* ***zbytkem*** *po dělení polynomu* f *uspořádanou s-ticí* F *a budeme ho označovat jako* $\overline{f}^F$*. Navíc, pokud* $g_i f_i \neq 0$*, pak* $\mathrm{multideg}(f) \geq \mathrm{multideg}(q_i f_i)$*.*

Důkaz této věty lze opět nalézt v [2, str. 64]. Součástí tohoto důkazu je i pseudokód algoritmu dělení, který je popsán v algoritmu 1.3.1.

Algoritmus 1.3.1 Algoritmus dělení v $k[x_1, ..., x_n]$

Vstup: polynomy $f_1, ..., f_s$, a f
Výstup: polynomy $q_1, ..., q_s$, a r

```
1:  function polynomialDivision(f_1, ..., f_s, f)
2:      q_1 ← 0, ..., q_s ← 0
3:      r ← 0
4:      p ← f
5:      while p ≠ 0 do
6:          i ← 1
7:          divisionoccured ← false
8:          while i ≤ s & divisionoccured = false do
9:              if LT(f_i) | LT(p) then
10:                 q_i ← q_i + LT(p)/LT(f_i)
11:                 p ← p − (LT(p)/LT(f_i))f_i
12:                 divisionoccured ← true
13:             else
14:                 i ← i + 1
15:         if divisionoccured = false then
16:             r ← r + LT(p)
17:             p ← p − LT(p)
18:     return q_1, ..., q_s, r
```

1.4 Gröbnerovy báze

V této sekci si již můžeme představit pojem Gröbnerova báze nějakého ideálu a poté si vysvětlit jejich význam při našich dalších výpočtech.

Definice 1.4.1. Nechť $I \subseteq k[x_1, ..., x_n]$ je ideál (jiný než $\{0\}$) a mějme nějaké monomiální uspořádání na $k[x_1, ..., x_n]$. Potom:

(i) Množinu vedoucích termů nenulových prvků z I budeme nazývat $\mathrm{LT}(I)$. Platí tedy, že

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

(ii) Ideál generovaný elementy z $\mathrm{LT}(I)$ pak budeme nazývat $\langle \mathrm{LT}(I) \rangle$.

Tvrzení 1.4.2. *Nechť $I \subseteq k[x_1, ..., x_n]$ je ideál jiný než $\{0\}$.*

(i) $\langle \mathrm{LT}(I) \rangle$ je monomiální ideál.

(ii) Existují $g_1, ..., g_t$, takové že $\langle \mathrm{LT}(I) \rangle = \langle \mathrm{LT}(g_1), ..., \mathrm{LT}(g_t) \rangle$.

Důkaz tohoto tvrzení je opět uveden v [2, str. 77].

Věta 1.4.3. ***Hilbertova věta o bázi.*** *Každý ideál $I \subseteq k[x_1, ..., x_n]$ má konečnou generující množinu. Tj. $I = \langle g_1, ..., g_t \rangle$, pro nějaké $g_1, ..., g_t \in I$.*

Důkaz je opět uveden v [2, str. 77].

Definice 1.4.4. Mějme monomiální uspořádání na okruhu polynomů $I \subseteq k[x_1, ..., x_n]$. Konečnou podmnožinu $G = \{g_1, ..., g_t\}$ ideálu $I \subseteq k[x_1, ..., x_n]$ různého od $\{0\}$ nazveme **Gröbnerova báze** (nebo **standardní báze**), pokud

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

Použitím konvence $\langle \emptyset \rangle = \{0\}$ zadefinujeme, že prázdná množina $\emptyset$ je Gröbnerovou bází triviálního ideálu $\{0\}$.

Tato definice by se také dala vyjádřit tak, že množina $\{g_1, ..., g_t\} \subseteq I$ je Gröbnerovou bází ideálu I právě tehdy, když vedoucí term kteréhokoli prvku z I je dělitelný nějakým prvkem z $\mathrm{LT}(g_i)$.

Důsledek 1.4.5. *Mějme monomiální uspořádání na $k[x_1, ..., x_n]$. Pak každý ideál $I \subseteq k[x_1, ..., x_n]$ má Gröbnerovu bázi. Navíc jakákoli Gröbnerova báze ideálu I je bází I.*

Důkaz je uveden v [2, str. 78].

Definice 1.4.6. Nechť $I \subseteq k[x_1, ..., x_n]$ je ideál. Označením $\mathbf{V}(I)$ budeme značit množinu:

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

I když nenulový ideál I vždy obsahuje nekonečně mnoho různých polynomů, tak množina $\mathbf{V}(I)$ může být definována konečnou množinou polynomiálních rovnic.

Tvrzení 1.4.7. *$\boldsymbol{V}(I)$ je afinní varieta. Speciálně, pokud $I = \langle f_1, ..., f_s \rangle$, pak $\boldsymbol{V}(I) = \boldsymbol{V}(f_1, ..., f_s)$.*

Nejdůležitějším důsledkem toho tvrzení je, že variety jsou určovány pomocí ideálů a jeho důkaz je uveden v [2, str. 81].

Tvrzení 1.4.8. *Nechť $I \subseteq k[x_1, ..., x_n]$ je ideál a $G = \{g_1, ..., g_t\}$ je Gröbnerova báze I. Pak pro dané $f \in k[x_1, ..., x_n]$ existuje jednoznačně určené $r \in k[x_1, ..., x_n]$ takové, které má následující vlastnosti:*

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

(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.

Důkaz tohoto tvrzení je uveden v [2, str. 83].

Důsledek 1.4.9. *Nechť $G = \{g_1, ..., g_t\}$ je Gröbnerova báze ideálu $I \subseteq k[x_1, ..., x_n]$ a nechť $f \in k[x_1, ..., x_n]$. Potom $f \in I$ právě tehdy, když zbytek po dělení polynomu f t-ticí polynomů z G je 0.*

Důkaz tohoto důsledku je opět uveden v [2, str. 84].

Definice 1.4.10. Nechť $f, g \in k[x_1, ..., x_n]$ jsou nenulové polynomy.

(i) Pokud $\mathrm{multideg}(f) = \alpha$ a $\mathrm{multideg}(g) = \beta$, pak nechť $\gamma = (\gamma_1, ..., \gamma_n)$, kde $\gamma_i = \max(\alpha_i, \beta_i)$ pro všechna i. Monom x^γ nazveme **nejmenším společným násobkem** $\mathrm{LM}(f)$ a $\mathrm{LM}(g)$ a budeme ho psát jako

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

(ii) Jako **S-polynom** f a g označíme polynom

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

Jako příklad uvedeme polynomy $f = 2x^2y^4 - 4x^3y^2$ a $g = x^8y + 2y^2$ z $\mathbb{R}[x, y]$ a uvažujme odstupňované lexikografické uspořádání. Nyní můžeme vypočítat:

$$\mathrm{S}(f, g) = \frac{x^8y^4}{2x^2y^4} \cdot f - \frac{x^8y^4}{x^8y} \cdot g =$$
$$= 1/2x^6 \cdot f - y^3 \cdot g = x^8y^4 - 2x^9y^2 - x^8y^4 - 2y^5 = -2x^9y^2 - 2y^5$$

Všimněme si, že S-polynom je zkonstruován právě tak, aby při jeho výpočtu došlo k odstranění vedoucích termů.

Věta 1.4.11. ***Buchbergerovo kritérium.*** *Nechť I je polynomiální ideál. Pak báze $G = \{g_1, ..., g_t\}$ ideálu I je jeho Gröbnerovou bází právě tehdy, když pro každou dvojici $i \neq j$ je zbytek po dělení S-polynomu $\mathrm{S}(g_i, g_j)$ t-ticí polynomů z G nula. Tj.*

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

Důkaz této věty je opět uveden v [2, str. 86].

Tímto jsme si ukázali nejdůležitější vlastnosti Gröbnerových bází. Díky Buchbergerovu kritériu jsme nyní schopni zjistit, zda daná báze ideálu je skutečně Gröbnerovou bází.

Toto kritérium je také základem tzv. Buchbergerova algoritmu, kterým se dá Gröbnerova báze daného ideálu vypočítat. Tento algoritmus byl poprvé představen v [3].

Algoritmus 1.4.1 Buchbergerův algoritmus na výpočet GB

Vstup: množina polynomů $F = (f_1, ..., f_s)$
Výstup: Gröbnerova báze $G = (g_1, ..., g_t)$ ideálu $I = \langle f_1, ..., f_s \rangle$

```
1:  function BuchbergerAlg(F)
2:      G ← F
3:      repeat
4:          G' ← G
5:          for každý pár {p, q}, p ≠ q v G' do
6:              r ← S(p, q) mod G'
7:              if r ≠ 0 then
8:                  G ← G ∪ {r}
9:      until G' = G
10:     return G
```

Pseudokód v algoritmu 1.4.1 byl převzat z [2, str. 91], kde je i dovysvětlen. Tím už se zde více zabývat nebudeme a raději se později zaměříme

na algoritmus F4, který budeme v práci dále využívat. Pro něj však musíme nyní zadefinovat ještě pár dalších vlastností Gröbnerových bází.

Definice 1.4.12. Redukovaná Gröbnerova báze polynomiálního ideálu I je taková Gröbnerova báze ideálu I, pro kterou platí:

(i) $\mathrm{LC}(p) = 1$ pro všechna $p \in G$.

(ii) Žádný monom z p neleží v $\langle \mathrm{LT}(G \setminus \{p\}) \rangle$ pro všechna $p \in G$.

Věta 1.4.13. *Nechť $I \neq \{0\}$ je polynomiální ideál. Pak při daném monomiálním uspořádání má ideál I redukovanou Gröbnerovu bázi, která je jedinečná.*

Důkaz této věty je uveden v [2, str. 93].

Definice 1.4.14. Mějme monomiální uspořádání a nechť $G = \{g_1, ..., g_t\} \subseteq k[x_1, ..., x_n]$. Pro polynom $f \in k[x_1, ..., x_n]$ řekneme, že f se **redukuje na nulu modulo** G, psáno $f \to_G 0$, pokud f má **standardní reprezentaci** ve tvaru

$$f = A_1 g_1 + ... + A_t g_t, A_i \in k[x_1, ..., x_n],$$

tj. kdykoli $A_i g_i \neq 0$, máme $\mathrm{multideg}(f) \geq \mathrm{multideg}(A_i g_i)$.

Lemma 1.4.15. *Nechť $G = \{g_1, ..., g_t\}$ je uspořádaná množina prvků z $k[x_1, ..., x_n]$ a mějme $f \in k[x_1, ..., x_n]$. Pak $\overline{f}^G = 0$ implikuje $f \to_G 0$, i když opačné tvrzení je všeobecně nepravdivé.*

Věta 1.4.16. *Báze $G = \{g_1, ..., g_t\}$ ideálu I je jeho Gröbnerovou bází právě tehdy, když $\mathrm{S}(g_i, g_j) \to_G 0$ pro každou dvojici $i \neq j$.*

Tato věta nám tedy říká, že báze ideálu I je jeho Gröbnerovou bází právě tehdy, když všechny jeho S-polynomy mají standardní reprezentaci. Důkazy obou výše uvedených tvrzení jsou uvedeny v [2, str. 105].

1.5 Algoritmus F4 pro výpočet Gröbnerových bází

Tento algoritmus byl poprvé představen v [4]. Přestože pojem F4 ve skutečnosti označuje celou skupinu algoritmů, my se zde pokusíme popsat a vysvětlit hlavní myšlenku, která je společná pro všechny jeho varianty.

Ta se od Buchbergerova algoritmu liší především ve výběru daných párů polynomů ke zpracování. Buchberger v [3] tvrdí, že pro získání správného

výsledku na pořadí zpracování jednotlivých párů nezáleží. Pokud nám však jde o efektivitu programu a rychlost jeho běhu, tak na tomto pořadí záležet začíná.

Například v [5] se uvádí, že nejefektivnější strategie výběru se zaměřuje na vedoucí termy. Pak ovšem nastává problém, když má více polynomů stejný vedoucí term.

Algoritmus F4 tento problém vyřešil tím, že k žádnému výběru pořadí nedochází, neboť se všechny dvojice zpracovávají zároveň. Tím se současně výrazně zvýšila efektivita výpočtu Gröbnerových bází. [4]

Protože v předchozích sekcích jsme při definování potřebných matematických konstrukcí vycházeli především z [2], budeme v tom pokračovat i nadále, abychom zachovali především stejnou notaci.

Nejprve si v algoritmu 1.5.1 uvedeme celý pseudokód pro F4. A poté se pokusíme vysvětlit jeho jednotlivé části.

Algoritmus 1.5.1 Základní myšlenka algoritmu F4

Vstup: množina polynomů $F = (f_1, ..., f_s)$
Výstup: Gröbnerova báze $G = (g_1, ..., g_t)$ ideálu $I = \langle f_1, ..., f_s \rangle$

1: **function** F4(F)
2: $G \leftarrow F$
3: $t \leftarrow s$
4: $B \leftarrow \{\{i, j\} \mid 1 \leq i < j \leq s\}$
5: **while** $B \neq \emptyset$ **do**
6: zvol $B' \neq \emptyset, B' \subseteq B$
7: $B \leftarrow B \setminus B'$
8: $L \leftarrow \bigcup_{\{i,j\} \in B'} \left\{ \frac{\text{lcm}(\text{LM}(f_i), \text{LM}(f_j))}{\text{LT}(f_i)} \cdot f_i, \frac{\text{lcm}(\text{LM}(f_i), \text{LM}(f_j))}{\text{LT}(f_j)} \cdot f_j \right\}$
9: $M \leftarrow \text{SpoctiM}(L, G)$
10: $N \leftarrow$ redukovaný odstupňovaný tvar matice M
11: $N^+ \leftarrow \{n \in \text{radky}(N) \mid \text{LM}(n) \notin \langle \text{LM}(\text{radky}(M)) \rangle\}$
12: **for** $n \in N^+$ **do**
13: $t \leftarrow t + 1$
14: $f_t \leftarrow$ polynomiální forma n
15: $G \leftarrow G \cup \{f_t\}$
16: $B \leftarrow B \cup \{\{i, t\} \mid 1 \leq i < t\}$
17: **return** G

Množina G je na začátku algoritmu inicializována všemi polynomy ze vstupní množiny F. V průběhu algoritmu se pak do ní přidávají další vygenerované polynomy, dokud se z ní nestane Gröbnerova báze ideálu ge-

nerovaného původní množinou F. Do proměnné t se přitom ukládá vždy aktuální počet polynomů v G.

B je množina, která obsahuje dvojice indexů $\{i, j\}$ takových polynomů z G, pro které ještě není ověřeno, zda

$$\mathrm{S}(g_i, g_j) \to_G 0. \tag{1.1}$$

Na začátku jsou to všechny vzájemné dvojice polynomů z G. Tuto podmínku musí dle věty 1.4.16 nakonec splnit všechny polynomy z G, aby G mohla být Gröbnerovou bázi.

Algoritmus se tedy ve smyčce **while** snaží množinu B postupně redukovat tím, že vždy vybere jeho určitou podmnožinu B', kterou z ní bude chtít odstranit. Výběr této podmnožiny se liší pro jednotlivé varianty algoritmu F4. Základním výběrem je tzv. normální selekční strategie, která je podrobně popsána v [4].

Nyní je však třeba do množiny G přidat "správné" polynomy tak, aby všechny polynomy určené indexy z B' splňovaly podmínku 1.1. Buchbergerův algoritmus (viz. 1.4.1) by nyní spočítal zbytky $\overline{\mathrm{S}(f_i, f_j)}^G$ pro každé $\{i, j\}$ z B' a ty přidal do G. Algoritmus F4 (1.5.1) však místo Buchbergerova kritéria (1.4.11) využívá větu 1.4.16 a počítá $\overline{\mathrm{S}(f_i, f_j)}$ definované pomocí rovnice

$$\mathrm{S}(f_i, f_j) - c_1 x^{\alpha(1)} f_{l_1} - c_2 x^{\alpha(2)} f_{l_2} - \ldots = \overline{\mathrm{S}(f_i, f_j)}. \tag{1.2}$$

Tyto polynomy jsou pak doplněny do G a tím získáme standardní reprezentaci $\mathrm{S}(f_i, f_j)$, čímž je podmínka z věty 1.4.16 splněna.

Významnou výhodou F4 (1.5.1) oproti Buchbergerovu algoritmu (1.4.1) je, že dokáže polynomy $\overline{\mathrm{S}(f_i, f_j)}$ pro všechny dvojice z B' spočítat zároveň. Nyní si tedy musíme ještě vysvětlit, jak přesně k tomu dojde.

Začneme vytvořením množiny L, která obsahuje prvky ve tvaru

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

pro každou dvojici $\{i, j\}$ z B'. Zde si můžeme všimnout, že rozdíl dvou prvků pro nějaký pár $\{i, j\}$ nám dá právě $\mathrm{S}(f_i, f_j)$ (viz. 1.4.10).

Množiny L a G jsou pak vstupem funkce *SpoctiM*, která vrací matici M. Abychom porozuměli, co tato matice reprezentuje, musíme si nejprve zadefinovat pár dalších pojmů.

Definice 1.5.1. Nechť $\mathrm{Mon}(f)$ je množina monomů obsažených v $f \in k[x_1, \ldots, x_n]$. Navíc pro množinu polynomů $K \subseteq k[x_1, \ldots, x_n]$ pak píšeme

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

Když si vezmeme nějakou množinu $H = \{f_1, ..., f_r\} \subseteq k[x_1, ..., x_n]$ a monomiální uspořádání $>$, pak $\mathrm{Mon}(H) = \{m_1, ..., m_s\}$.

Definice 1.5.2. Nechť

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

je uspořádaná s-tice monomů z $\mathrm{Mon}(H)$ taková, že $m_{i_1} > ... > m_{i_s}$. Potom M je matice typu $r \times s$ taková, že

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

Jinak řečeno 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 $>$. [6]

Dále ještě zbývá uvést, co znamená matice v odstupňovaném tvaru. Následující definice byly převzaty z [7, str. 111, 118].

Definice 1.5.3. První nenulový prvek zleva v nenulovém řádku matice se nazývá **pivot**. Říkáme, že matice je v **odstupňovaném tvaru**, pokud jsou splněny následující podmínky:

(i) Všechny nulové řádky matice leží pod těmi nenulovými.

(ii) Pivot na všech řádcích matice je vždy více vpravo než všechny pivoty řádků nad ním.

Definice 1.5.4. Říkáme, že matice je v **redukovaném odstupňovaném tvaru**, pokud jsou splněny následující podmínky:

(i) Všechny nulové řádky matice leží pod těmi nenulovými.

(ii) Pivot na všech řádcích matice je vždy více vpravo než všechny pivoty řádků nad ním.

(iii) Každý pivot je roven 1.

(iv) Každý prvek matice, který leží ve sloupci nad pivotem, je roven 0.

Nyní si již můžeme dopodrobna popsat, jak funguje funkce *SpoctiM*, jejíž pseudokód uvádíme v algoritmu 1.5.2.

Algoritmus 1.5.2 Funkce SpočtiM(L,G) z alg. F4

Vstup: $L, G = (f_1, ..., f_t)$
Výstup: matice M

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

Jak bylo popsáno výše, matice M nám zde reprezentuje množinu H. Tato množina však musí splňovat následující dvě podmínky:

(i) $L \subseteq H$,

(ii) kdykoli nastane, že x^β je monom nějakého polynomu $f \in H$ (tj. $x^\beta \in \mathrm{Mon}(H)$) a zároveň existuje $f_l \in G$ takové, že $\mathrm{LM}(f_l) \mid x^\beta$, pak H obsahuje prvek $x^\alpha f_l$ takový, že $\mathrm{LM}(x^\alpha f_l) = x^\beta$.

Funkce *SpoctiM* (1.5.2) začíná tím, že množinu H inicializuje všemi polynomy z množiny L. Tím se rovnou splní podmínka (i) pro množinu H.

Poté se v cyklu zpracovávají všechny monomy z množiny $\mathrm{Mon}(H)$. Zpracované monomy se přidají do proměnné množiny *done* a cyklus běží tak dlouho, dokud se *done* a $\mathrm{Mon}(H)$ nerovnají. Tyto množiny lze snadno porovnávat, protože jsou obě sestupně seřazené vzhledem k $>$.

Vzhledem k tomu, že množina H po původní inicializaci obsahuje oba polynomy

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

pro každou dvojici $\{i, j\}$ z B', dá se odvodit, že x^β je roven nějakému vedoucímu monomu z H. Tím je tedy podmínka (ii) množiny H splněna pro všechny monomy z $\mathrm{LM}(H)$, a proto je můžeme přiřadit do množiny *done* ještě před začátkem samotného cyklu.

V cyklu **while** se pak ke zpracování vždy vybírá největší x^β z ještě nezpracovaných monomů v $\mathrm{Mon}(H)$. Pokud existuje $f_l \in G$ takové, že $\mathrm{LM}(f_l) \mid x^\beta$, pak do množiny H přidáme polynom $\frac{x^\beta}{\mathrm{LM}(f_l)} \cdot f_l = x^\alpha f_l$ s vedoucím monomem x^β. Dělá se to proto, že monom x^β se nesmí objevit na pravé straně rovnice z (1.2), a proto musí být na levé straně něčím eliminován. A tím pro dané x^β splníme podmínku (ii).

Společně s množinou H upravujeme vždy i množinu $\mathrm{Mon}(H)$, aby si v každém kroku algoritmu navzájem odpovídaly. Cyklus skončí v momentě, kdy všechny monomy z $\mathrm{Mon}(H)$ budou zpracovány a žádné další už v této množině nebudou přibývat.

Posledním krokem funkce *SpoctiM* (1.5.2) je pak konstrukce matice koeficientů M odpovídající polynomům v množině H, jak jsme si ukázali výše v 1.5.2. Tato matice je pak z funkce vrácena jako její výsledek.

V dalším kroku algoritmu F4 (1.5.1) se pak spočítá matice N, jako redukovaný odstupňovaný tvar matice M, jejíž tvar je uveden v 1.5.4. Její výpočet nám současně vyprodukuje rovnice v požadovaném tvaru (1.2).

Do N^+ pak algoritmus 1.5.1 přiřadí pouze ty polynomy (řádky) z N, jejichž vedoucí termy nejsou dělitelné vedoucími termy žádného polynomu z M. Takto vzniklé polynomy pak odpovídají tvaru $\overline{\mathrm{S}(f_i, f_j)}$ z (1.2).

Posledním bodem vnějšího **while** cyklu algoritmu 1.5.1 je pak **for** cyklus, ve kterém se polynomy určené jednotlivými řádky matice N^+ přidají do množiny G a odpovídající dvojice indexů do množiny B.

Poté, co se množina B zcela vyprázdní, máme zaručeno, že G splňuje podmínku z 1.4.16. Konkrétní příklad algoritmu F4 v akci lze nalézt v [2, str. 573].

Kapitola 2

Advanced Encryption Standard

V této kapitole zkráceně vysvětlíme, jak funguje bloková šifra *Advanced Encryption Standard* (zkráceně AES) a její zjednodušené varianty. A poté si popíšeme systém polynomiálních rovnic, kterým se dají tyto šifry namodelovat.

Nakonec si představíme skripty od Ing. Marka Bielika [1], které umožňují tento systém rovnic vypočítat pomocí mnoha různých metod a experimentálně porovnávat jejich efektivitu.

2.1 Historie AES

Nejprve si krátce řekneme něco k historii této šifry. Původně byla jednou z nejrozšířenějších blokových šifer šifra DES. Se svou délkou bloku 64 bitů však s postupným nárůstem výpočetních možností začínala být čím dál méně bezpečná.

Poté, co útoky hrubou silou dokázaly, že DES již není dostačující, se v roce 1997 americký standardizační úřad rozhodl vypsat výběrové řízení na novou blokovou šifru - tedy AES. Na rozdíl od DES má AES délku bloku 128 bitů a podporuje celkem tři délky tajného klíče: 128, 192 a 256 bitů.

Výběrové řízení nakonec vyhrála šifra Rijndael od autorů Joan Daemen a Vincent Rijmen v roce 2001 byla publikována jako oficiální standard v [8].

Autoři vítězné šifry v [9] uvádějí, že jediný rozdíl mezi původní šifrou Rijndael a oficiálním standardem AES je podporovaný rozsah délky bloku a tajného klíče. Zatímco AES má délku bloku pevně zafixovanou na 128 bitů a délka klíče může nabývat tří konkrétních hodnot, tak pro Rijndael mohu obě délky nabývat všech hodnot mezi 128 a 256 dělitelných 32.

2.2 Stuktura AES

V této části si představíme, jak algoritmus šifry AES funguje. Nejprve si zadefinujeme důležité pojmy, které budeme v tomto textu nadále používat. Poté si vysvětlíme, jakým způsobem probíhá prodloužení tajného klíče a nakonec popíšeme samotný algoritmus šifrování pomocí AES.

2.2.1 Důležité pojmy

Začneme tím, že si vysvětlíme, co je to vlastně *blok*. Jako blok budeme označovat jednoduše sekvenci bitů určité délky (v našem případě konkrétně 128), která se v šifrovacím algoritmu zpracovává najednou. Algoritmus si tedy vždy vezme jeden blok dat otevřeného textu a po dokončení vrátí jeden blok jemu příslušného šifrového textu.

Dalším důležitým pojmem je pro nás proměnná *state*. Tato pracovní proměnná v sobě udržuje důležitá data v průběhu celého algoritmu ve formě dvojdimenzionálního pole o velikosti 16 bytů (což je přesně 1 blok). Toto pole je rozděleno na 4 řádky a 4 sloupce, takže jednotlivé byty mohou být odkazované pomocí proměnné $s_{r,c}$ nebo $s_{i,j}$, kde r/i je index řádku a c/j index sloupce. [8]

Každý blok se pak může rozdělit ještě na jednotlivé části o velikosti 32 bitů (4 byty) nazývané *words* (česky slova). Každý sloupec proměnné *state* pak obsahuje právě jedno *slovo* daného bloku dat.

Dále je nutno říci, že algoritmus pomocí zadaných funkcí provádí transformace dat, jež se opakují v cyklu tolikrát, kolik algoritmus obsahuje tzv. *rund*. Počet rund se odvíjí od velikosti tajného klíče. V této práci se zaměříme pouze na variantu s délkou klíče 128 bitů, takže algoritmus bude obsahovat 10 rund.

Nakonec ještě musíme zadefinovat binární operaci násobení polynomů v $\mathrm{GF}(2^8)$, pro které platí, že výsledek násobení se ještě musí zredukovat modulo *ireducibilní polynom* stupně 8. Polynom je ireducibilní právě tehdy, když je dělitelný pouze 1 nebo sám sebou. Operaci násobení polynomů budeme označovat $\bullet$.

V algoritmu AES se za konkrétní ireducibilní polynom pro redukci vybere polynom

$$m(x) = x^8 + x^4 + x^3 + x + 1. \tag{2.1}$$

Z toho pak vyplývá, že algoritmus pracuje s polynomy z tělesa definovaného jako $\mathrm{GF}(2)[x] \setminus \langle m(x) \rangle$. Pokud tedy budeme v kontextu s AES dále hovořit o $\mathrm{GF}(2^8)$, budeme tím mít na mysli právě toto těleso.

2.2.2 Expanze klíče

Prvním krokem je úprava tajného klíče pomocí funkce *KeyExpantion* tak, aby obsahoval určitá data pro každou rundu algoritmu AES (+ jednou úplně na začátku).

Původní klíč byl dlouhý 128 bitů - tedy 16 bytů, které se mohou rozdělit do 4 *slov* (words). Funkce *KeyExpantion* z těchto 4 slov vyrobí pole o velikosti 44 slov, ze kterého se potom v každé rundě postupně použijí vždy 4 slova. Přitom první 4 slova navíc se použijí ještě před první rundou a ty obsahují právě původní tajný klíč.

Celý postup expanze klíče je popsán pomocí pseudokódu v algoritmu 2.2.1.

Algoritmus 2.2.1 Pseudokód funkce KeyExpantion

Vstup: tajný klíč (key[16B])
Výstup: expandovaný tajný klíč ($EK[4 \cdot (10+1)\text{B}]$)

1: **function** KeyExpantion(key)
2: word EK[44B], word $temp$[4B]
3: **for** $i \leftarrow 0$ až 3 **do**
4: $EK[44] \leftarrow \text{word}(key[4i], key[4i+1], key[4i+2], key[4i+3])$
5: **for** $i \leftarrow 4$ až 43 **do**
6: $temp \leftarrow EK[i-1]$
7: **if** $i \mod 4 = 0$ **then**
8: $temp = \text{SubWord}(\text{RotWord}(temp)) \oplus Rcon[i/4]$
9: $EK[i] \leftarrow EK[i-4] \oplus temp$
10: **return** EK

Jak již bylo řečeno, na začátku se do prvních 4 slov expandovaného klíče prostě na kopírují data celého tajného klíče. Každé další slovo expandovaného klíče pak vznikne jako xor slova předchozího a toho, které se nachází o čtyři pozice dříve.

Navíc pro slova, která se nachází na pozici dělitelné čtyřmi, projde toto slovo před xorováním ještě transformací, kterou zařídí funkce *RotWord*(word) a *SubWord*(word) a konstanta *Rcon*, které si nyní popíšeme.

Funkce *RotWord* dostane na vstupu určité slovo $w = [a_0, a_1, a_2, a_3]$ a to cyklicky zarotuje tak, že výstupem je pak slovo ve tvaru $[a_1, a_2, a_3, a_0]$. Funkce *SubWord* pak toto slovo vezme a aplikuje na něj substituci definovanou pomocí tabulky *S-box*, o které budeme ještě mluvit.

K výsledku substituce se pak ještě přixoruje příslušná část konstanty *Rcon*[40]. Konkrétní část *Rcon*[i] obsahuje data definovaná jako $Rcon[i] = [x_{16}^{i-1}, 00_{16}, 00_{16}, 00_{16}]$, kde x_{16}^{i-1} jsou mocniny $02_{16} \in \text{GF}(2^8)$.

2.2.3 Popis šifrování

Nyní se již můžeme podívat na samotný algoritmus šifrování otevřeného textu. V algoritmu 2.2.2 uvedeme pseudokód celkového algoritmu šifry AES převzatého z [8] a poté se pokusíme velice stručně popsat jeho jednotlivé části.

Algoritmus 2.2.2 Pseudokód hlavního algoritmu pro šifru AES

Vstup: otevřený text (OT[16B]), expandovaný tajný klíč (EK[4·(10+1)B])
Výstup: šifrový text (CT[16B])

```
1:  function AES(OT, key)
2:      state ← OT
3:      state ← AddRoundKey(state, EK[0 : 3])
4:      for runda ← 1 až 9 do
5:          state ← SubBytes(state)
6:          state ← ShiftRows(state)
7:          state ← MixColumns(state)
8:          state ← AddRoundKey(state, EK[4·runda : 4·(runda+1)−1])
9:      state ← SubBytes(state)
10:     state ← ShiftRows(state)
11:     state ← AddRoundKey(state, EK[40 : 43])
12:     return state = CT
```

Na začátku se do proměnné *state* nakopíruje jeden blok dat otevřeného textu a poté se provede první zavolání funkce *AddRoundKey*. Tato funkce má za úkol k proměnné *state* jednoduše přixorovat příslušná slova expandovaného klíče. Na začátku algoritmu jsou to právě první 4 slova jež odpovídají původnímu tajnému klíči.

Poté už následuje cyklus, ve kterém se pro všechny rundy vykonají postupně funkce *SubBytes*, *ShiftRows*, *MixColumns* a *AddRoundKey*. Pouze v poslední rundě už se funkce *MixColumns* neprovádí.

Funkci *AddRoundKey* jsme si už vysvětlili a nyní se tedy podíváme na ty ostatní.

Funkce *SubBytes* je jedinou nelineární transformací celého algoritmu. Jedná se o substituci jednotlivých bytů proměnné *state* pomocí substituční tabulky zvané *S-box*, uvedené na obrázku 2.1.

		y														
		0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
x	0	63	7c	77	7b	f2	6b	6f	c5	30	01	67	2b	fe	d7	ab	76
	1	ca	82	c9	7d	fa	59	47	f0	ad	d4	a2	af	9c	a4	72	c0
	2	b7	fd	93	26	36	3f	f7	cc	34	a5	e5	f1	71	d8	31	15
	3	04	c7	23	c3	18	96	05	9a	07	12	80	e2	eb	27	b2	75
	4	09	83	2c	1a	1b	6e	5a	a0	52	3b	d6	b3	29	e3	2f	84
	5	53	d1	00	ed	20	fc	b1	5b	6a	cb	be	39	4a	4c	58	cf
	6	d0	ef	aa	fb	43	4d	33	85	45	f9	02	7f	50	3c	9f	a8
	7	51	a3	40	8f	92	9d	38	f5	bc	b6	da	21	10	ff	f3	d2
	8	cd	0c	13	ec	5f	97	44	17	c4	a7	7e	3d	64	5d	19	73
	9	60	81	4f	dc	22	2a	90	88	46	ee	b8	14	de	5e	0b	db
	a	e0	32	3a	0a	49	06	24	5c	c2	d3	ac	62	91	95	e4	79
	b	e7	c8	37	6d	8d	d5	4e	a9	6c	56	f4	ea	65	7a	ae	08
	c	ba	78	25	2e	1c	a6	b4	c6	e8	dd	74	1f	4b	bd	8b	8a
	d	70	3e	b5	66	48	03	f6	0e	61	35	57	b9	86	c1	1d	9e
	e	e1	f8	98	11	69	d9	8e	94	9b	1e	87	e9	ce	55	28	df
	f	8c	a1	89	0d	bf	e6	42	68	41	99	2d	0f	b0	54	bb	16

Obrázek 2.1: Substituční tabulka S-box pro byty xy_{16} [8]

Tato tabulka byla vytvořena pomocí kombinace následujících dvou transformací:

(i) Vezmeme multiplikativní inverzi v GF(2^8) (prvek 00_{16} je mapován sám na sebe).

(ii) Aplikujeme následující afinní transformaci nad GF(2^8):

$$b'_i = b_i \oplus b_{(i+4) \mod 8} \oplus b_{(i+5) \mod 8} \oplus b_{(i+6) \mod 8} \oplus b_{(i+7) \mod 8} \oplus c_i$$

pro $0 \leq i < 8$, kde b_i je i-tý bit příslušného bytu a c_i je i-tý bit bytu $c = 63_{16}$.

Tato transformace může být také popsána pomocí maticového tvaru tímto způsobem:

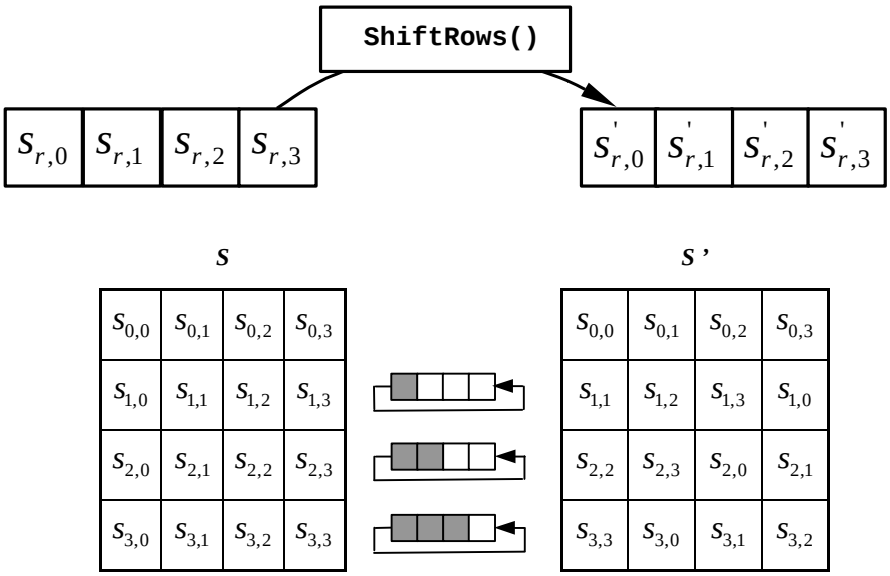
$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \\ b'_4 \\ b'_5 \\ b'_6 \\ b'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \\ b_4 \\ b_5 \\ b_6 \\ b_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{2.2}$$

Funkce *ShiftRows* pracuje s jednotlivými řádky proměnné *state*. Její transformace je definována jako:

$$s'_{r,c} = s_{r,(r+c) \mod 4} \text{ pro } 0 < r < 4 \text{ a } 0 \leq c < 4.$$

Jinak řečeno, každý řádek je cyklicky zarotován doleva o počet bytů určený indexem daného řádku, pokud indexujeme od 0. To znamená, že první řádek se neposune vůbec, druhý se posune o jeden byte, třetí o dva a čtvrtý se posune o celkem tři byty.

Na obrázku 2.2 je tento proces znázorněn graficky.



Obrázek 2.2: Schéma transformace prováděné funkcí ShifRows [8]

Jako poslední nám tedy zbývá vysvětlit funkci *MixColumns*. Tato transformace se pro změnu provádí na jednotlivých sloupcích proměnné *state*. Každý sloupec je zde vnímán jako 4-termový polynom z GF(2^8) a je vynásoben modulo $x^4 + 1$ s konstantím polynomem $a(x) = 03_{16}x^3 + 01_{16}x^2 + 01_{16}x + 02_{16}$.

Tento postup by se dal také vyjádřit pomocí maticového násobení $s'(x) = a(x) \otimes s(x)$ vyjádřeného takto:

$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \\ s'_{2,c} \\ s'_{3,c} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \\ s_{2,c} \\ s_{3,c} \end{bmatrix} \text{ pro } 0 \leq c < 4. \quad (2.3)$$

Výsledkem tohoto násobení jsou potom jednotlivé byty daného sloupce v tomto tvaru:

$$s'_{0,c} = (02_{16} \bullet s_{0,c}) \oplus (03_{16} \bullet s_{1,c}) \oplus s_{2,c} \oplus s_{3,c}$$

$$s'_{1,c} = s_{0,c} \oplus (02_{16} \bullet s_{1,c}) \oplus (03_{16} \bullet s_{2,c}) \oplus s_{3,c}$$

$$s'_{2,c} = s_{0,c} \oplus s_{1,c} \oplus (02_{16} \bullet s_{2,c}) \oplus (03_{16} \bullet s_{3,c})$$

$$s'_{3,c} = (03_{16} \bullet s_{0,c}) \oplus s_{1,c} \oplus s_{2,c} \oplus (02_{16} \bullet s_{3,c})$$

2.3 Zjednodušené varianty AES

Vzhledem k výpočetní složitosti řešení problému nalezení tajného klíče šifry AES bylo zapotřebí pro účely kryptoanalýzy vyvinout takovou šifru, která by měla podobné algebraické vlastnosti jako AES, ovšem její analýza by nebyla tak složitá.

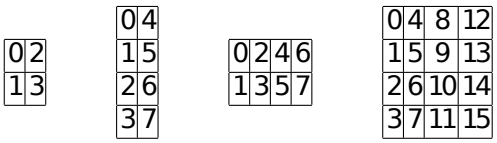
Pánové Cid, Murphy a Robshaw vytvořili ve své práci *Small Scale Variants of the AES* [10] celou skupinu takových šifer, které budeme nazývat *zjednodušenými variantami AES*. Každá z nich se dá charakterizovat čtyřmi parametry *n, r, c,* a *e*, jejichž určitou kombinací pak dostaneme jednu určitou variantu šifry nazvanou *SR(n,r,c,e)* nebo *SR*(n,r,c,e)* (jejich rozdíl si popíšeme později).

Nyní si vysvětlíme, co znamenají jednotlivé parametry:

- n - počet rund prováděných šifrovacím algoritmem, $1 \leq n \leq 10$
- r - počet řádků proměnné *state*, $r = 1, 2, 4$
- c - počet sloupců proměnné *state*, $c = 1, 2, 4$
- e - počet bitů udávající velikost *slova* (*word*), $e = 4, 8$

Z jednotlivých výše uvedených parametrů lze pak vypočítat velikost *bloku* dat, kterou budou jednotlivé varianty šifry používat. Tuto velikost *bloku* si můžeme definovat jako $r \cdot c \cdot e$ bitů, které jsou reprezentovány pomocí pole o velikosti $r \cdot c$ slov.

Na obrázku 2.3 uvádíme rozložení slov v proměnné *state* pro jednotlivé varianty šifry, se kterými budeme později pracovat. Zleva je to: SR$(n, 2, 2, e)$, SR$(n, 4, 2, e)$, SR$(n, 2, 4, e)$ a SR$(n, 4, 4, e)$.



Obrázek 2.3: Rozložení slov v proměnné *state* ve zjednodušených variantách AES [10]

Pro parametr e je ještě nutné podotknout, že jednotlivé šifry pracují s daty z tělesa $\mathrm{GF}(2^e)$ a pro každé e tedy musí být definován ireducibilní polynom.

Polynom pro $\mathrm{GF}(2^8)$ jsme si již zadefinovali v (2.1) při popisu šifry AES. Pro všechny varianty $\mathrm{SR}(n, r, c, 8)$ tedy platí, že pracují s polynomy z tělesa definovaného jako $\mathrm{GF}(2)[x] \setminus \langle x^8 + x^4 + x^3 + x + 1 \rangle$.

Pro varianty $\mathrm{SR}(n, r, c, 4)$ pak vybereme ireducibilní polynom $p(x) = x^4 + x + 1$. Těleso $\mathrm{GF}(2^4)$ pro nás tedy v tomto kontextu bude znamenat $\mathrm{GF}(2)[x] \setminus \langle x^4 + x + 1 \rangle$.

Při původním algoritmus AES se každá runda skládá z výše definovaných funkcí *SubBytes*, *ShiftRows*, *MixColumns* a *AddRoundKey*, kromě poslední rundy, kde *MixColumns* chybí. To samé platí i pro zjednodušené varianty šifry $\mathrm{SR}^*(n, r, c, e)$, které používají lehce zjednodušené varianty těchto funkcí tak, aby odpovídaly parametrům jejich nastavení.

Ty samé zjednodušené funkce pak používají i varianty $\mathrm{SR}(n, r, c, e)$. Ovšem pro ty platí, že poslední runda se od předchozích neliší a obsahuje funkce všechny, včetně *MixColumns*.

Z popisu všech výše uvedených parametrů nyní vyplývá, že plnou verzi algoritmu AES můžeme nazvat variantou $\mathrm{SR}^*(10, 4, 4, 8)$. Vzhledem k tomu, že při šifrování otevřeného textu stejným tajným klíčem se šifrové texty z verzí $\mathrm{SR}^*(n, r, c, e)$ a $\mathrm{SR}(n, r, c, e)$ dají navzájem převádět pomocí afinního mapování, můžeme v naší práci používat pouze verze $\mathrm{SR}(n, r, c, e)$.

Nyní si ještě popíšeme rozdíly v jednotlivých transformačních funkcích pro jejich zjednodušené varianty. Začneme funkcí*SubBytes*, která opět provádí substituci jednotlivých bytů proměnné *state* pomocí substituční tabulky *S-box*. Pro $e = 8$ tato tabulka vzniká tak, jak jsme si definovali výše pro plný AES a je znázorněna na obrázku 2.1.

Pro $e = 4$ je pak menší tabulka znázorněna na obrázku 2.4 a je vytvořena pomocí podobných transformací jako (2.2), které si popíšeme nyní.

(i) Vezmeme multiplikativní inverzi v $\mathrm{GF}(2^4)$ (prvek 0_{16} je mapován sám na sebe).

(ii) Aplikujeme následující afinní transformaci nad GF(2^4):

$$\begin{bmatrix} b'_0 \\ b'_1 \\ b'_2 \\ b'_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \\ b_3 \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \tag{2.4}$$

S-Box over GF (2^4)															
Input	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Output	6	B	5	4	2	E	7	A	9	D	F	C	3	1	0	8

Obrázek 2.4: Substituční tabulka S-box pro zjednodušené varianty AES s $e = 4$ [10]

Funkce *ShiftRows* pracuje nezávisle na počtu řádků i sloupců, takže ji můžeme zadefinovat stejně jako pro nezjednodušený AES. Každý řádek je cyklicky zarotován doleva o počet bytů určený indexem daného řádku, pokud indexujeme od 0.

Funkce *MixColumns* se pro $c = 4$ opět chová stejně jako při plném AES, kde jsme si tuto lineární transformaci ukázali pomocí maticového násobení v (2.3). Pro $c = 2$ pak tento vzorec přepíšeme na:

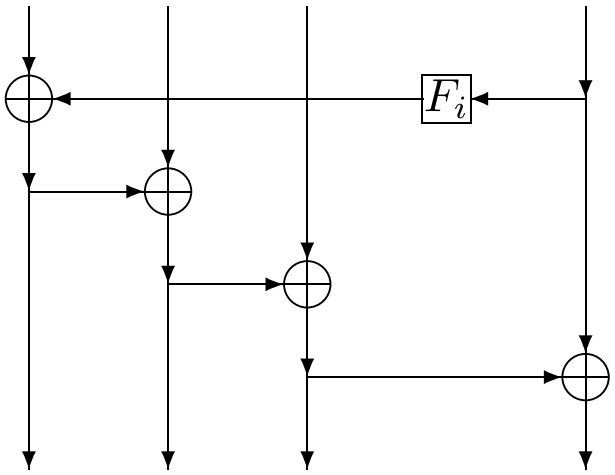
$$\begin{bmatrix} s'_{0,c} \\ s'_{1,c} \end{bmatrix} = \begin{bmatrix} 03 & 02 \\ 02 & 03 \end{bmatrix} \begin{bmatrix} s_{0,c} \\ s_{1,c} \end{bmatrix} \text{ pro } 0 \leq c < 2. \tag{2.5}$$

Ještě je nutno podotknout, že pro $e = 4$ a $e = 8$ se používá vždy příslušné těleso GF(2^e) a jeho ireducibilní polynom.

Funkce *AddRoundKey* pak pracuje pro všechny varianty stejně. Na začátku algoritmu a při každé rundě prostě přixoruje příslušnou část expandovaného klíče k proměnné *state*. Pouze její velikost se odvíjí od zadaných parametrů.

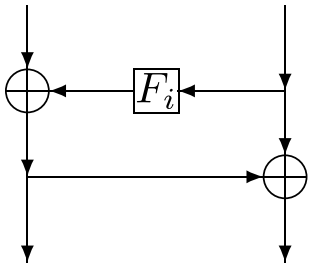
Rozdíl mezi variantami však nastává ve funkci *KeyExpantion* právě při vyrábění expandovaného klíče z toho, který máme zadaný. Samotná délka původního klíče již není fixovaná na 128 bitů, ale počítá se ze zadaných parametrů jako $(r \cdot c) \cdot e$. Délka expandovaného klíče se pak spočítá jako $(c) \cdot (n + 1) \cdot e$.

Ve 2. kapitole jsme si v sekci 2.2.2 podrobně popsali algoritmus funkce *KeyExpantion* pro plný AES a vysvětlili jeho jednotlivé prvky. Tento algoritmus by se dal jednoduše znázornit pomocí obrázku 2.5. Na něm je schéma opakující se pro každou rundu, kde F_i je transformace zařízená pomocí funkcí *RotWord* a *SubWord* a rundovní konstanty.



Obrázek 2.5: Schéma algoritmu pro KeyExpantion pro plný AES [10]

Pro zjednodušené formy se pak počet rund odvíjí od parametru n a velikost daných bloků dat od r, c a e. Nebudeme si zde popisovat kód pro každou variantu, protože princip je stále stejný. Místo toho uvedeme na obrázku 2.6 schéma jedné rundy pro verzi s $c = 2$.



Obrázek 2.6: Schéma algoritmu pro KeyExpantion pro zjednodušené varianty AES s $c = 2$ [10]

2.4 Převod na systém polynomiálních rovnic

Systém rovnic pro šifru AES se skládá ze dvou částí. Jedna část se získává z transformací probíhajících během expanze klíče a druhá pak při samotném šifrování. Zatímco systém pro expanzi klíče závisí pouze na proměnných tohoto klíče, část systému pro šifrování už zahrnuje i proměnné jednotlivých otevřených a šifrových textů. Díky tomu můžeme pro každou dvojici

otevřeného a šifrového textu vygenerovat jiný polynominální systém i při použití stejného tajného klíče. [11]

Existuje několik způsobů, jakými lze polynomiální systém rovnic pro šifru AES vytvořit, které jsou popsány například v [10], [11] nebo [12]. My se v této práci zaměříme na systém, který vytvořil a popsal Ing. Marek Bielik pro svou diplomovou práci *Algebraic Cryptanalysis of Small Scale Variants of the AES* [13], na kterou budeme dále navazovat.

Systémy rovnic pro všechny verze AES se modelují stejným způsobem. Pro názornost zde však uvedeme konkrétní příklad šifry $\mathrm{SR}(n, 2, 2, 4)$.

2.4.1 Kvadratické rovnice

Ze všeho nejdříve se zaměříme na jedinou nelineární transformaci celého algoritmu, která nastává při aplikaci tabulky S-box, kvůli jejím vlastnostem popsaných v bodě (i) v popisu vytváření zmenšené tabulky S-box.

Pokud si vezmeme polynom $b \in \mathrm{GF}(2^e)$ jako vstup a $c \in \mathrm{GF}(2^e)$ jako výstup aplikace S-boxu, víme, že $bc = 1$, pokud však nenastane situace $b = 0$. Pravděpodobnost této situace je však minimální viz [11], a pokud by k tomu došlo, vyhneme se tomuto problému tím, že vybereme jiná vstupní data. Dále je nutno podotknout, že pokud budeme nadále mluvit o násobení, máme na mysli operaci polynomiálního násobení zadefinovanou jako $\bullet$.

Nyní si tedy můžeme zadefinovat první 4 rovnice, jež popisují výsledek operace polynomiálního násobení v $\mathrm{GF}(2)[x] \setminus \langle x^4 + x + 1 \rangle$ pro $b \bullet c = 1$, kde koeficienty b_i a c_i jsou z $\mathrm{GF}(2)$:

$$\begin{aligned} 1 &= b_0c_0 \oplus b_3c_1 \oplus b_2c_2 \oplus b_1c_3 \\ 0 &= b_1c_0 \oplus b_0c_1 \oplus b_3c_2 \oplus b_2c_3 \oplus b_3c_1 \oplus b_2c_2 \oplus b_1c_3 \\ 0 &= b_2c_0 \oplus b_1c_1 \oplus b_0c_2 \oplus b_3c_3 \oplus b_3c_2 \oplus b_2c_3 \\ 0 &= b_3c_0 \oplus b_2c_1 \oplus b_1c_2 \oplus b_0c_3 \oplus b_3c_3 \end{aligned} \tag{2.6}$$

Pokud bychom pracovali v $\mathrm{GF}(2^8)$, vzniklo by nám podobným způsobem rovnic celkem 8.

Vztah $bc = 1$ nám však může pomoci získat i další rovnice. Můžeme například celý vzorec vynásobit polynomem b nebo c a získat tak vztahy $bc^2 = c$ a $b^2c = b$ a z nich pak rovnice $bc^2 + c = 0$ a $b^2c + b = 0$.

Abychom získali kvadratické rovnice ze vztahu $bc^2 + c = 0$, musíme nejprve spočítat c^2. Protože v (2.6) jsme již dostali výsledek operace bc, můžeme v něm nyní nahradit b_i za c_i a tím získáme následující koeficienty:

$$\begin{aligned} c_0^2 &= c_0 \oplus c_2 \\ c_1^2 &= c_2 \\ c_2^2 &= c_1 \oplus c_3 \\ c_3^2 &= c_3 \end{aligned} \tag{2.7}$$

Tyto koeficienty pak můžeme dosadit do vztahu $bc^2 + c = 0$ a výsledkem budou následující 4 kvadratické rovnice:

$$\begin{aligned} 0 &= b_0c_0 \oplus b_0c_2 \oplus b_3c_2 \oplus b_2c_1 \oplus b_2c_3 \oplus b_1c_3 \oplus c_1 \\ 0 &= b_1c_0 \oplus b_1c_2 \oplus b_0c_2 \oplus b_3c_1 \oplus b_3c_3 \oplus b_3c_2 \oplus b_2c_1 \oplus b_1c_3 \oplus c_1 \\ 0 &= b_2c_0 \oplus b_2c_2 \oplus b_1c_2 \oplus b_0c_1 \oplus b_0c_3 \oplus b_3c_1 \oplus b_2c_3 \oplus c_2 \\ 0 &= b_3c_0 \oplus b_3c_2 \oplus b_2c_2 \oplus b_1c_1 \oplus b_1c_3 \oplus b_0c_3 \oplus b_3c_3 \oplus c_3 \end{aligned}$$

Stejným postupem můžeme získat i rovnice pro vztah $b^2c + b = 0$ a dohromady tedy získáme 8 dalších rovnic do našeho systému. A opět zde můžeme podotknout, že pokud bychom pracovali v GF(2^8), vzniklo by nám takovým způsobem rovnic celkem 16.

V dalším kroku pak můžeme původní vztah vynásobit určitou mocninou jedné z jeho proměnných. Nejvýhodnější pro nás bude mocnina třetí, protože při použití druhé mocniny by výsledek obsahoval rovnice kubické a ne jenom kvadratické.

Vezmeme si tedy vztah $bc^4 = c3$ a opět ho upravíme na $bc^4 + c^3 = 0$. Koeficienty pro c^4 získáme jako výsledek násobení c^2c^2, kdy c^2 jsem si již spočítali v (2.7) a pomocí substituce do prvních rovnic (2.6) pak po všech úpravách získáme rovnice:

$$\begin{aligned} 0 &= b_3c_3 \oplus b_3c_1 \oplus b_2c_3 \oplus b_2c_2 \oplus b_1c_3 \oplus b_0c_3 \oplus b_0c_2 \oplus b_0c_1 \oplus b_0c_0 \\ &\quad \oplus c_3c_1 \oplus c_2c_1 \oplus c_2c_0 \oplus c_0 \\ 0 &= b_3c_2 \oplus b_3c_1 \oplus b_2c_2 \oplus b_1c_2 \oplus b_1c_1 \oplus b_1c_0 \oplus b_0c_3 \oplus b_0c_1 \oplus c_3c_2 \\ &\quad \oplus c_2c_0 \oplus c_1c_0 \oplus c_3 \\ 0 &= b_3c_2 \oplus b_2c_2 \oplus b_2c_1 \oplus b_2c_0 \oplus b_1c_3 \oplus b_1c_1 \oplus b_0c_3 \oplus b_0c_2 \oplus c_3c_2 \\ &\quad \oplus c_3c_1 \oplus c_3c_0 \oplus c_2c_1 \oplus c_2c_0 \oplus c_1c_0 \oplus c_2 \\ 0 &= b_3c_2 \oplus b_3c_1 \oplus b_3c_0 \oplus b_2c_3 \oplus b_2c_1 \oplus b_1c_3 \oplus b_1c_2 \oplus b_0c_3 \oplus c_3c_2 \\ &\quad \oplus c_3c_2 \oplus c_3c_1 \oplus c_3 \oplus c_2 \oplus c_1 \end{aligned}$$

Stejným postupem opět získáme i rovnice pro vztah $b^4c + b^3 = 0$ a dohromady tedy získáme dalších 8 rovnic do našeho systému. Celkem tedy nakonec máme 20 kvadratických rovnic pro GF(2^4) a 40 pro GF(2^8). Jak je uvedeno v [11] a [13], pro náš systém nebudeme potřebovat všech 20 rovnic, ale postačí nám prvních 12.

2.4.2 Lineární rovnice

Další rovnice už budeme konstruovat pouze z lineárních transformací, takže i tyto rovnice budou lineární. Začneme tím, že si představíme rovnice, které dokončí aplikaci tabulky S-box.

Tyto rovnice vycházejí z druhého kroku konstrukce tabulky uvedeného v bodě (ii), při popisu zmenšené tabulky S-box, a mohou se přímo vyjádřit pomocí výrazu z (2.4), kde jako vstup do tohoto výrazu vezmeme polynom $c \in \mathrm{GF}(2^e)$, který je výstupem transformace popsané v předchozí sekci 2.4.1. Když nyní takto vzniklé 4 lineární rovnice zkombinujeme s 12 kvadratickými rovnicemi z předchozí sekce, získáme 16 rovnic které dohromady přesně popisují tabulku pro S-box.

A nyní už se zaměříme na modelování jednotlivých funkcí celého algoritmu AES. Pro zjednodušení budeme operovat s daty, která popisují celé dvojrozměrné pole proměnné *state*, které je pro variantu šifry SR$(n, 2, 2, 4)$ ukázáno na obrázku 2.3 úplně vlevo.

Pro funkci *SubBytes* si zavedeme matici L, která vychází z matice použité pro výrobu S-boxu ve výrazu (2.4), takovýmto způsobem:

$$L_b = \begin{pmatrix} 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 \end{pmatrix} \longrightarrow L = \begin{pmatrix} L_b & 0 & 0 & 0 \\ 0 & L_b & 0 & 0 \\ 0 & 0 & L_b & 0 \\ 0 & 0 & 0 & L_b \end{pmatrix}$$

Zároveň s tím si rozšíříme i konstantní vektor $(0, 1, 1, 0)^T = 6_{16}$ z (2.4) na $\mathbf{v} = (6_{16}, 6_{16}, 6_{16}, 6_{16})$, abychom mohli pracovat s celým *state* a ne jen s jednotlivými byty. Dále si dle notace použité v [13] označíme vstupní vektor funkce *SubBytes* jako $\mathbf{b}$ a výstupní jako $\mathbf{b^{-1}}$ (protože obsahuje inverzní prvky z $\mathbf{b}$). Každý prvek těchto vektorů se skládá ze 4 koeficientů polynomů b a c definovaných v předchozí sekci 2.4.1, takže každý prvek pak přináší do našeho systému 12 kvadratických rovnic.

Pro funkci *ShiftRows* si pak zavedeme matici R, která bude reprezentovat rotaci jednotlivých řádků:

$$R = \begin{pmatrix} I_4 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_4 \\ 0 & 0 & I_4 & 0 \\ 0 & I_4 & 0 & 0 \end{pmatrix}$$

kde I_4 je jednotková matice velikosti 4×4.

Pro znázornění funkce *MixColumns* si nejprve přepíšeme operaci polynomiálního násobení $b \bullet c$ z (2.6) do maticového taru:

$$\begin{pmatrix} b_0 & b_3 & b_2 & b_1 \\ b_1 & b_0 \oplus b_3 & b_3 \oplus b_2 & b_2 \oplus b_1 \\ b_2 & b_1 & b_0 \oplus b_3 & b_3 \oplus b_2 \\ b_3 & b_2 & b_1 & b_0 \oplus b_3 \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{pmatrix} \tag{2.8}$$

Pokud nyní za jednotlivé bity b_i v matici z (2.8) dosadíme binární hodnoty koeficientů 03_{16} a 02_{16} z (2.5), dostaneme následující dvě matice:

$$M_{03} = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \text{ a } M_{02} = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

Matice M, která popisuje celou funkci *MixColumns*, pak bude vypadat takto:

$$M = \begin{pmatrix} M_{03} & M_{02} & 0 & 0 \\ M_{02} & M_{03} & 0 & 0 \\ 0 & 0 & M_{03} & M_{02} \\ 0 & 0 & M_{02} & M_{03} \end{pmatrix}$$

S takto zadefinovanými pojmy můžeme nyní vyjádřit jednu celou rundu $0 < i \leq n$ šifry $\mathrm{SR}(n, 2, 2, 4)$ pomocí tohoto výrazu:

$$\mathbf{b_i} = MRL\mathbf{b_{i-1}^{-1}} + \mathbf{k_i} + \mathbf{v}$$

kde $\mathbf{k_i}$ je vektor obsahující 16 bitů reprezentujících proměnné v dané části expandovaného klíče pro rundu i.

Z tohoto výrazu dostaneme pro každou rundu 16 lineárních rovnic. K těm navíc připočítáme 12 kvadratických pro každý prvek z $\mathbf{b_{i-1}^{-1}}$, takže dohromady máme v našem systému celkem 64 rovnic pro každou rundu algoritmu. [13]

Na začátku dostaneme $\mathbf{b_0}$ tak, že vezmeme hodnoty otevřeného textu a přidáme k nim proměnné inicializační části expandovaného klíče, což jsou ve skutečnosti přímo proměnné tajného klíče, které se snažíme najít. Výraz $\mathbf{b_0} = OT + \mathbf{k_0}$ nám tedy přidá do systému ještě dalších 16 rovnic před první rundou.

2.4.3 Rovnice pro expanzi klíče

Označme si $k_i = (k_{i,0}, k_{i,1}, k_{i,2}, k_{i,3})^T \in \mathrm{GF}(2^4)^4$ za danou část tajného klíče pro rundu i. Tato část je pak dle [10] pro každou rundu i definována jako:

$$\begin{pmatrix} k_{i,2q} \\ k_{i,2q+1} \end{pmatrix} = \begin{pmatrix} L(s_0) \\ L(s_1) \end{pmatrix} + \begin{pmatrix} 6_{16} \\ 6_{16} \end{pmatrix} + \begin{pmatrix} \kappa_i \\ 0 \end{pmatrix} + \sum_{t=0}^{q} \begin{pmatrix} k_{i-1,2t} \\ k_{i-1,2t+1} \end{pmatrix}$$

pro $0 \leq q < 2$, kde $s_0 = k_{i-1,3}^{-1}, s_1 = k_{i-1,2}^{-1}$ a κ_i je rundovní konstanta z $\mathrm{GF}(2^4)$. Z tohoto výrazu dostaneme 16 lineárních rovnic pro každou rundu.

K tomu navíc musíme přidat ještě další kvadratické rovnice. Při generování částí klíče pro každou rundu se totiž ještě vždy dvakrát aplikuje S-box. Pro každou rundu tedy do našeho systému přidáme ještě $2 \cdot 12$ kvadratických rovnic, které jsme si definovali v sekci 2.4.1.

Pro každou rundu tedy do našeho systému přidáme celkem 40 lineárních a kvadratických rovnic, které popisují expandovaný tajný klíč. [13]

2.5 Skripty na měření efektivity výpočtu tajného klíče

Pan Bielik pro svou diplomovou práci [13] vytvořil také sadu skriptů [1], které pro náhodně vygenerovaný tajný klíč zašifrují zadaný počet otevřených textů do šifrových textů a potom z nich namodelují systém polynomiálních rovnic.

Takto vygenerované polynomy jsou potom dále zpracovávány různými funkcemi a nakonec předány softwaru Magma [14]. Ten vypočte jejich Gröbnerovu bázi a z ní je potom schopen zpětně vypočítat původní tajný klíč. Nebo jsou tyto polynomy předělány do vhodného tvaru a předány softwaru CryptoMiniSat [15], který klíč vypočítá jako řešení soustavy logických formulí.

Při tom dochází k pečlivému měření časové a paměťové náročnosti běhu celého výpočtu. Díky tomu jsme schopni porovnat efektivitu výpočtu tajného klíče šifry AES při použití různých druhů zpracování daného polynomiálního systému.

2.5.1 Popis skriptů

Celý kód je rozdělen do tří skriptů psaných v jazyce Python, verze minimálně 3.8. Soubor *config.py* obsahuje, jak jeho název napovídá, především konfigurační proměnné a další nastavení potřebná pro správné měření a vytištění času běhu jednotlivých funkcí.

Dále je zde soubor *saes.py* popisující dvě hlavní třídy: *AES* a *PolynomialSystem*. Třída *AES* obsahuje funkce a metody, které mají za úkol vygenerovat tajný klíč a otevřené texty a zašifrovat je pomocí zadané varianty šifry AES. Z těchto dat pak také vypočítá požadovaný systém polynomiálních rovnic. Ten se poté předá funkcím ze třídy *PolynomialSystem*, které ho podle uživatelem zadaných parametrů vyřeší požadovaným způsobem. Celý systém, včetně všech konkrétních polynomů, je v kódu uložen pomocí struktur z balíčku *PolyBoRi* ([16]), jež umožňuje s polynomy dále poměrně efektivně pracovat.

Celý algoritmus se spouští pomocí výchozího skriptu *experiments.py*. Ten má za úkol především naparsovat uživatelem zadané argumenty a podle nich pak vybrat příslušné funkce ze skriptu *saes.py*, které budou zadaný problém řešit.

Uživatel má na výběr z celé škály přepínačů a jejich argumentů. V první řadě je třeba navolit jaká varianta bude pro šifrování použita. To zařizují přepínače *-n, -r, -c, -e*, ke kterým uživatel vždy zadá požadované číslo, aby se tak vytvořila požadovaná verze šifry *SR(n,r,c,e)*.

Dále si může uživatel pomocí přepínače *--sat* zvolit, zda se polynomiální systém bude řešit pomocí softwaru Magma nebo CryptoMiniSat. Pokud uživatel tento přepínač nezadá, použije se defaultní nastavení pro Magmu.

Posledním důležitým obecným přepínačem je *--num*. K němu uživatel zadá číslo, jež udává počet otevřených textů, které skript vygeneruje a poté zašifruje. Kolik párů těchto textů bude mít k dispozici, tolik pak bude moci vytvořit polynomiálních systémů.

K němu je možné přidat ještě přepínač *--sim*, který udává, že vygenerované otevřené texty se mají navzájem lišit pouze v jednom bitu. Budou si tedy navzájem dost podobné.

Kombinací dalších přepínačů a jejich argumentů pak uživatel nastavuje jednotlivé typy zpracování napočítaných polynomů. Ty si uvedeme v další sekci.

2.5.2 Jednotlivé způsoby zpracování polynomů

Jak přesně jednotlivé typy zpracování fungují uvádí Ing. Bielik ve svém publikovaném článku [17]. Proto se tím zde nebudeme zabývat moc dopodrobna a raději si ukážeme, jak jednotlivé typy spustit, aby si případný uživatel mohl provést konkrétní experimenty sám.

Jako první typ zvolil Ing. Bielik výpočet polynomiálního systému, který obsahuje kromě proměnných tajného klíče ještě další pomocné proměnné. Díky tomu není závislý na hodnotách jednotlivých otevřených a šifrových textů, a proto nemá smysl jich v tomto případě generovat více. Tento algoritmus se dá spustit pomocí přepínače *--aux*.

V následující metodě se pak do polynomiálního systému přidávají i hodnoty daných textů a tím se původní pomocné proměnné eliminují. Výsledné polynomy tedy obsahují pouze proměnné tajného klíče. Tato metoda je ve skriptech uvažována jako základní, a proto pro její spuštění nejsou potřeba žádné zvláštní argumenty.

Jako další pak uvažujeme rozšíření této metody přidáním více párů otevřených a šifrových textů pomocí přepínače *--num*. Ke každému páru se vypočítá jeden polynomiální systém a jejich kombinace pak může zrychlit výpočet Gröbnerovy báze.

V tomto případě pak už rychlost běhu celého algoritmu záleží i na tom, kolik takových párů vygenerujeme. Najít optimální počet není triviální, protože více polynomů sice může urychlit výpočet tajného klíče pomocí Gröbnerovy báze, ale zároveň déle trvá než se všechny systémy vytvoří.

Při následujícím typu zpracování se opět vygeneruje několik polynomiálních systémů za pomoci více párů textů. V tomto případě se však zvolí jeden jako základní a ze všech ostatních pak vznikne jedna množina.

Ke každému polynomu ze základního systému se pak bude v druhé množině hledat takový polynom, který je mu nejpodobnější (což znamená, že má nejvíce stejných monomů). Takto nalezené dvojice se pak navzájem vyxorují. Výsledný polynom je potom mnohem kratší, protože skripty pracují nad GF(2) a všechny stejné monomy se navzájem vyruší.

Pro tuto verzi algoritmu se skripty spouští opět s přepínačem *--num* a číslem pro požadovaný počet párů otevřených a šifrových textů. Navíc se ještě musí přidat přepínač *--red*, který zajistí výše popsanou redukci na jeden konečný polynomiální systém.

Další možností vylepšení základního algoritmu je pak použití útoku hrubou silou na několik předem určených proměnných tajného klíče předtím, než se začnou počítat Gröbnerovy báze či řešit logické formule. K určení tohoto typu algoritmu slouží přepínač *--guess* se svým argumentem, který udává počet proměnných, které se mají předem spočítat.

Které proměnné se budou počítat pomocí útoku hrubou silou nám pak určují další přepínače. Algoritmus pracuje tak, že si nejdříve spočítá četnost jednotlivých proměnných ve všech polynomech a v defaultním případě pak nastaví výběr těch nejčastěji se vyskytujících proměnných. Uživatel si však může pomocí přepínače *--rev* zvolit, že chce vybrat ty nejméně časté anebo pomocí přepínače *--rnd* nastavit výběr náhodný.

Tato metoda se dá navíc kombinovat s libovolnou z výše uvedených metod, čímž se efektivita běhu programu může ještě celkově zvýšit.

Kapitola 3

Návrh a implementace redukce systému rovnic

Hlavním úkolem této práce je vyzkoušet další způsoby zpracování napočítaných polynomiálních systémů tak, aby výpočet tajného klíče byl ještě efektivnější.

Budeme uvažovat pouze metody výpočtu tajného klíče s využitím Gröbnerovy báze, která je spočítána pomocí algoritmu F4 (1.5.1) softwarem Magma [14].

Zaměříme se tentokrát na metody, při kterých se vygeneruje velké množství různých otevřených textů a k nim příslušných šifrových textů, takže vznikne velké množství polynomů. Ty se potom budeme snažit pomocí různých algoritmů zredukovat tak, aby výpočet Gröbnerovy báze a tajného klíče byl co nejrychlejší.

3.1 Algoritmus PAM

Jako první jsme se rozhodli vyzkoušet využití metody shlukové analýzy, neboli "klastrování". Jeho princip spočívá v rozdělení příslušných dat do skupin tak, že v každé skupině jsou vždy taková data, která jsou si navzájem co nejpodobnější. [18]

V našem případě příslušnými daty myslíme jednotlivé polynomy ze všech napočítaných polynomiálních systémů a kritériem podobnosti bude jejich vzájemná vzdálenost. Tu určíme jako počet monomů, které mají dané polynomy rozdílné, což se dá spočítat jako počet monomů polynomu vzniklého jejich vzájemným xorem.

Nyní si ukážeme, že vzájemný xor polynomů splňuje všechny axiomy definice vzdálenosti tak, jak jsou definovány v [19].

(i) $d(p,q) = 0$ právě tehdy když $p = q$ - Protože naše polynomy jsou definovány v GF(2), tak se při xorování navzájem vyruší všechny monomy, které jsou obsažené v obou polynomech p, q a ve výsledku zůstanou pouze ty rozdílné. Aby tedy výsledný polynom mohl být nulový, musí p obsahovat stejné monomy jako q a tudíž $p = q$.

(ii) $d(p,q) = d(q,p)$ - Přímo z definice operace xor vyplývá, že je komutativní. [20]

(iii) $d(p,q) \leq d(p,r)+d(r,q)$ - Důkaz této vlastnosti by se dal vyčíst z pravdivostní tabulky operace xor. My ji zde však uvádět nebudeme. V [18] se totiž uvádí, že tento axiom nemusí být pro potřeby klastrování splněn.

3.1.1 Popis implementace algoritmu

Jako nejvhodnější klastrovací algoritmus pro naše data jsme zvolili algoritmus *Partitioning Around Medoids*, zkráceně PAM, protože ten je založen na hledání k reprezentativních objektů, tzv. *mediodů*, které pochází ze zadané množiny dat [18]. Jinak řečeno klastry (skupiny dat) jsou v tomto případě tvořeny okolo center, která jsou vždy vybírána pouze ze zadané množiny.

Tím se liší od častěji využívaného algoritmu *K-Means*, který také rozděluje data do skupin okolo určitých center, ovšem tato centra se vypočítají jako optimální středy všech dat z dané skupiny a v původní množině se vůbec nemusí vyskytovat. V našem případě by však nastal problém, pokud bychom pak polynom reprezentující toto centrum chtěli použít pro výpočet Gröbnerovy báze, protože by nemusel být součástí ideálu, pro který GB počítáme.

Na začátku algoritmu jsou mediody vybrány náhodně a poté se spočítá vzdálenost všech prvků z množiny ke každému z nich. Jednotlivé prvky se pak přiřadí k tomu mediodu, ke kterému mají vzdálenost nejmenší.

Poté se spočítá tzv. *celková cena* tohoto přiřazení, která se rovná součtu vzdáleností všech dat k jejich příslušným mediodům.

Dále následuje **while** cyklus, ve kterém se zkusí vybrat jiné mediody. V naší implementaci však není výběr nových mediodů vůbec náhodný.

Algoritmus pracuje v cyklu a pro každý mediod vykoná následující příkazy. Pro každý prvek, který přísluší danému mediodu, si přepočítá vzdálenosti tak, jakoby on byl jeho centrem a poté spočítá novou celkovou cenu.

Pokud ta je lepší než cena původní, aktualizuje se daný mediod právě na tento prvek. Ovšem pokud je cena horší, výpočet se zahodí a neděje se nic.

Vnější **while** cyklus pak pokračuje tak dlouho, dokud se mediody aktualizují. A když doběhne, dostaneme požadovaný počet klastrů okolo příslušných mediodů, čímž se algoritmus PAM ukončí.

Náš výpočet však stále pokračuje, protože požadovaným výstupem této metody zpracování polynomů nejsou celé klastry, ale pouze jeden zástupce z každého z nich. Místo toho, abychom vybrali jeden náhodný polynom, jsme se rozhodli, že výslednou soustavu ještě trochu zoptimalizujeme.

Z každého klastru proto vybereme dva polynomy, které jsou si navzájem nejpodobnější a jako zástupce tohoto klastru pak budeme brát výsledek jejich vzájemného xoru. Díky tomu pak budou vybrané polynomy kratší, než všechny původně vygenerované polynomy.

3.2 Algoritmus LSH

Jako další typ zpracování polynomů jsme se rozhodli využít principu algoritmů pro řešení problému nalezení *nejbližšího souseda*, který v [21] definován takto:

Definice 3.2.1. Mějme prvek p a cílem je najít takové $x \in \{x_1, ..., x_n\}$, pro které platí, že $d(p, x)$ je nejmenší ze všech $\{x_1, ..., x_n\}$. Kde $d(p, x)$ je jejich vzájemná vzdálenost.

Dnes již existují poměrně efektivní algoritmy na řešení tohoto problému pro data s nižším počtem dimenzí. Pro vícerozměrná data je však lepší použít algoritmy pro hledání *přibližně nejbližšího souseda*. [21]

V našem případě hledání nejpodobnějších polynomů jsme si vzdálenost d zadefinovali výše jako počet monomů jejich xoru. Pohybujeme se tedy v mnohadimenzionálním prostoru, a proto zvolíme spíše tento přístup.

Pokud se nám totiž podaří nalézt co nejpodobnější polynomy z celé vygenerované množiny, pak jejich xorem získáme opět mnohem jednodušší polynomy, které můžeme použít pro výpočet GB.

Pro naši implementaci jsme si vybrali skupinu algoritmů s názvem *Locality Sensitive Hashing*, zkráceně LSH, a to konkrétně verzi prezentovanou v [22]. Celý tento algoritmus se skládá z několika částí a ty si nyní popíšeme.

3.2.1 Popis implementace algoritmu

Na začátku je třeba vytvořit množiny všech monomů, které jednotlivé polynomy obsahují a z těchto množin se pak vytvoří jedna, která obsahuje

monomy ze všech polynomů. Z této množiny se pak vygeneruje slovník, který pro každý monom obsahuje jeho pořadové číslo.

V další části jednotlivé polynomy převedeme na tzv. *řídké binární vektory*. Každý prvek tohoto vektoru nyní reprezentuje právě jeden monom ze slovníku všech monomů. Pokud daný polynom určitý monom obsahuje, je na jeho přiděleném místě v binárním vektoru 1 a jinak jsou tam 0.

Dále už následuje první proces hashování, který převede řídké vektory na husté, kterým budeme říkat *signatury*. Tyto signatury mají opět pro všechny polynomy stejnou velikost. Ta se odvíjí od velikosti slovníku, přitom je však řádově menší.

Pro každý polynom se vytvoří 1 signatura tak, že pro každý jednotlivý prvek této signatury se vytvoří náhodná permutace čísel velikosti slovníku. V té permutaci se pak vyhledá, na jakém pořadovém místě se nachází číslo 1 a zjišťuje se, zda na stejném místě se v příslušném řídkém vektoru nachází 1. Pokud ano, uloží se číslo 1 z permutace jako jeden prvek signatury. Pokud ne, hledá se na jakém místě v permutaci je číslo 2 a opět se toto místo porovnává s řídkým vektorem. Pokud tam je 1, zapíše se jako prvek signatury číslo 2 a pokud ne opakuje se tento postup pro 3, 4 a tak dále.

V posledním kroku algoritmu se pak hledají podobné signatury, které nám určí tzv. *páry kandidátů* na podobné polynomy. Pokud bychom však hledali podobnost celých signatur, znamenalo by, to že hledáme pouze téměř stejné polynomy a kandidátů by se našlo velice málo.

Proto algoritmus nejprve rozdělí signatury ještě do několika podčástí, které se znovu zahashují, ovšem každá zvlášť. Pokud se pak pro dva různé polynomy objeví stejná hash alespoň pro nějakou část, označí se tyto dva polynomy za kandidáty na podobný pár.

Pro každý nalezený pár kandidátů se pak spočítá jejich vzájemná vzdálenost a polynom vzniklý jejich xorem. Tyto polynomy se pak podle vzdálenosti seřadí od nejmenší po největší a příslušný počet těch nejmenších se pak použije pro výpočet Gröbnerovy báze.

3.3 Algoritmus odstranění nejvýznamnějšího monomu

Jako poslední typ zpracování polynomů jsme se rozhodli vyzkoušet metodu, při které z velkého množství vygenerovaných polynomů vybereme pro výpočet Gröbnerovy báze právě ty, jejichž vedoucí monom, při použití obráceného odstupňovaného lexikografického uspořádání, má co nejmenší stupeň. S takovými polynomy by měl totiž algoritmus F4 (1.5.1) praco-

vat rychleji, protože ve funkci *SpoctiM* (popsanou v algoritmu 1.5.2) se v cyklu zpracovávají všechny monomy ze všech polynomů od největšího po nejmenší. Pokud bychom tedy vzali polynomy s co nejmenšími vedoucími monomy, bude se jich zpracovávat méně.

Navíc, pokud dva různé polynomy budou mít vedoucí monom stejný, můžeme je navzájem vyxorovat. Výsledný polynom pak bude mít stupeň vedoucího monomu ještě menší a tím bude pro výpočet výhodnější.

3.3.1 Popis implementace algoritmu

Pro tento algoritmus neexistuje žádný obecný postup ani jméno, a proto zde prostě popíšeme jeho implementaci v našich skriptech.

Nejprve bylo zapotřebí najít vedoucí monom a spočítat jeho stupeň pro každý polynom. V našich skriptech jsou polynomy implementovány pomocí struktur z balíčku *PolyBoRi* [16]. Zde jsou přímo naimplementovány funkce *lead()* a *deg()*, které tuto funkcionalitu zařizují.

Problém je však v tom, že tyto funkce operují s monomy řazenými pomocí lexikografického uspořádání, kdežto pro algoritmus F4 je výhodnější řadit monomy v obráceném odstupňovaném lexikografickém uspořádání, jak doporučuje jeho autor v [4]. Bylo tedy zapotřebí kód pro nalezení vedoucího monomu v tomto uspořádání vytvořit.

Pro všechny polynomy se jako vedoucí monom označí na začátku ten první a pak se v cyklu spočítá stupeň každého monomu. Pokud ten je větší, než stupeň původního vedoucího monomu, tak se vedoucí monom aktualizuje. Pokud jsou stejné, musí ještě nastat porovnání lexikograficky poslední proměnné daných monomů a za vedoucí monom se označí ten, jehož poslední proměnná je lexikograficky menší, aby se splnila podmínka obráceného odstupňovaného uspořádání.

Poté se všechny polynomy seřadí do pole podle velikosti stupně jejich vedoucího monomu od nejmenšího po největší a postupně se ke každému z nich hledá další polynom se stejným vedoucímu monomem. Když se taková dvojice najde, její xor se vybere do množiny pro výpočet Gröbnerovy báze a oba polynomy se z původního pole pro vyhledávání dvojic vyřadí.

Tento postup se opakuje tak dlouho, dokud nedostaneme požadovaný počet polynomů pro výpočet Gröbnerovy báze. Pokud bychom neměli dostatečný počet dvojic se stejným vedoucím monomem, doplní se tento výběr jednotlivými původně vygenerovanými polynomy.

Při šifrování první a druhé rundy většiny variant šifry AES však tato situace nenastává. Pokud bychom v budoucnu chtěli počítat i rundy vyšší, dal by se tento algoritmus ještě vylepší tím, že pokud dojdou dvojice se

stejným vedoucím monomem, mohly by se některé polynomy vynásobit určitou proměnou tak, aby další dvojice vznikly.

3.4 Implementace celkového rozšíření

Abychom mohli tyto druhy zpracování polynomu použít v našich experimentech, museli jsme upravit původní skripty z diplomové práce Ing. Bielika [1] tak, že jsme přidali do třídy *PolynomialSystem* metodu *ps_reduce_ml* a několik nových přepínačů.

Ze všeho nejdříve jsme museli zařídit, abychom při našich experimentech mohli provádět jednotlivá měření pro různé typy zpracování polynomů na stejných datech. Proto jsme přidali přepínače *--fstore, --fload* a *--fname* a příslušně upravili i kód.

Pokud uživatel zadá přepínač *--fstore*, znamená to, že náhodně vygenerovaný tajný klíč a všechny otevřené texty se mají vyexportovat do souboru specifikovaného pomocí přepínače *--fname*. Naopak, pokud bude zadán přepínač *--fload*, tak se klíč ani otevřené texty generovat nebudou a místo toho se načtou ze souboru, který je opět zadán jako argument přepínače *--fname*.

Dále jsme přidali přepínač *--ml*, který zajistí, že se vygenerované polynomiální systémy budou zpracovávat pomocí naší nové metody *ps_reduce_ml*. Zároveň s ním se pak může zadat i přepínač *--mlmeth*, který má za úkol vybrat příslušný algoritmus zpracování polynomů pomocí výběru jednoho z argumentů *"PAM","LHS","MSM"*.

Posledním přidaným přepínačem je pak *--psets*, který má za argument přirozené číslo. Toto číslo je pak v kódu použito pro výpočet celkového počtu polynomů, ze kterých se bude počítat Gröbnerova báze.

V defaultním nastavení má hodnotu 1, což znamená, že počet polynomů je roven přesně počtu bitů tajného klíče. Jiná čísla pak vždy udávají, že počet polynonů bude roven danému násobku bitů klíče.

Nyní se již zaměříme na metodu *ps_reduce_ml*, která se skládá z několika částí. Na začátku se metoda rozvětví podle algoritmu, který byl zadán v argumentu přepínače *--mlmeth*. Pro každý algoritmus se tedy vykonává jiný kód, který polynomy příslušně upraví. Podrobnosti k jednotlivým algoritmům jsou popsány výše.

Na konci metody se pak z daných polynomů vyrobí textové příkazy takovým způsobem, že později je bude možno předat softwaru Magma a ten na jejich základě spočítá Gröbnerovu bázi pro výpočet tajného klíče.

Kapitola 4

Experimenty a měření

Všechny experimenty byly prováděny na stroji s operačním systémem *Ubuntu 20.04.4 LTS*, který běží na dvou procesorech *Intel Xeon Gold 6136* obsahujících každý 12 jader. Celkem má tedy stroj k dispozici 24 jader a 768 GB RAM.

Cílem těchto experimentů pak bylo pro všechny tři algoritmy popsané v kapitole 3 najít ideální počet polynomiálních systémů, které mají skripty vygenerovat. A to tak, aby jejich redukcí pak vznikly polynomy, které budou pro výpočet Gröbnerovy báze co nejlepší a tím byl celý proces nalezení tajného klíče co nejefektivnější.

Pro každou ze zvolených zjednodušených variant šifry AES jsme provedli skupinu měření, při kterých jsme měnili parametr *num*, jež udává počet párů otevřených a šifrových textů, ze kterých se generují jednotlivé polynomiální systémy.

Protože délka výpočtu tajného klíče silně závisí na těchto vstupních datech, zopakovali jsme každé měření v průměru 10–20krát pro různé sady dat. Ze všech pokusů jsme potom spočítali průměrné hodnoty a jejich odchylky, abychom měli co nejpřesnější představu o skutečných výsledcích.

Pro každou šifru jsme pak vygenerovali grafy průměrného času generování polynomů a výpočtu tajného klíče v závislosti na hodnotě parametru *num*. Protože těchto grafů je hodně, nebudeme je zde uvádět všechny. Místo toho z každé kombinace vybereme ten nejlepší výsledek a jeho hodnoty si pro každou šifru vypíšeme do tabulky.

V každé tabulce se pak budou nacházet tyto záznamy:

- Šifra - Zvolená varianta šifry
- KB - Počet bitů klíče
- PS - Počet vygenerovaných polynomiálních systémů
- VP - Počet polynomů vybraných pro výpočet GB
- MON - Průměrný počet monomů v polynomech
- ČPP - Průměrný čas přípravy polynomů v sekundách
- ČVK - Průměrný čas výpočtu klíče v sekundách
- ČVC - Průměrný čas celého výpočtu v sekundách

(4.1)

Občas se stalo, že výpočet nedoběhl, což mohlo mít 2 příčiny. Buď to se objevila chyba ve výpočtu, která způsobila v našich skriptech výjimku, a výpočet se ukončil. Tuto výjimku většinou způsobila interní chyba v programu Magma.

Anebo trval výpočet déle než předem definovaný maximální čas výpočtu. Ten jsme stanovili různý v závislosti na metodě a složitosti šifry. V průměru se pohybovala jeho hodnota okolo 4 hodin. Nedokončené pokusy jsme potom z naší statistiky úplně vyloučili.

4.1 Referenční řešení

Ze všeho nejdříve jsme zahájili výpočty pro tzv. referenční řešení. Při nich jsme vygenerovali náhodná data, která jsme vyexportovali pro další použití. Tato data se skládají z binárních hodnot otevřených textů a tajného klíče, ze kterých se pak vypočítá šifrový text a požadované polynomy. Zároveň jsme měřili, jak dlouho trvá výpočet tajného klíče pro různý počet vygenerovaných polynomů bez použití jakékoliv redukce u zvolených šifer.

V [17] bylo poukázáno, že výpočet klíče může fungovat lépe, když je polynomů více, než kolik má tajný klíč (a tedy i otevřené a šifrové texty) bitů, což nastavuje $num = 1$. Proto jsme začali naše experimenty až od $num = 2$.

Výběr nejlepší hodnoty byl v tomto případě jednoduchý. Jelikož zde nedochází k žádné vhodné redukci polynomů, tak čím více jich vygenerujeme, tím delší je celkový čas výpočtu, protože se musí zpracovat více polynomů.

V tabulce 4.1 jsou tedy většinou vypsány hodnoty prvního běhu pro každou variantu šifry, které doběhly v přijatelném čase.

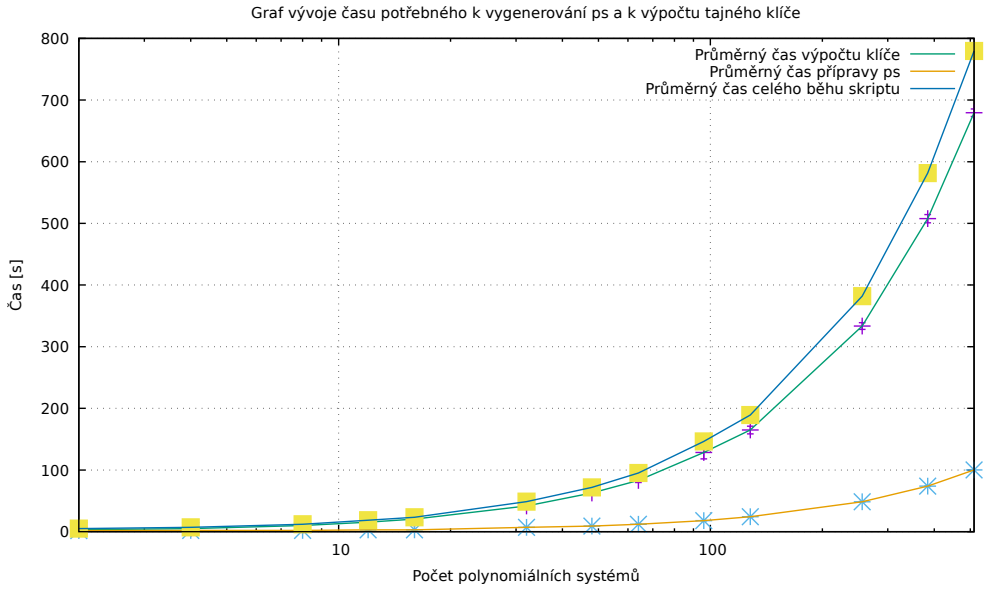
Tabulka 4.1: Naměřené časy běhu pro nejlepší konfigurace bez použití redukce - referenční řešení

Šifra	KB $[bity]$	PS	VP	MON	ČPP $[s]$	ČVK $[s]$	ČVC $[s]$
SR(1,2,2,4)	16	2	16	20	< 1	< 1	1
SR(2,2,2,4)	16	2	16	2451	2	3	5
SR(3,2,2,4)	16	2	16	32762	7	793	800
SR(1,4,2,4)	32	2	32	36	1	< 1	1
SR(2,4,2,4)	32	2	32	33142	5	—	—
SR(1,2,4,4)	32	2	32	23	1	< 1	1
SR(2,2,4,4)	32	2	32	6689	3	—	—
SR(1,4,4,4)	64	2	64	40	3	< 1	3
SR(1,2,2,8)	32	2	32	316	7	< 1	7
SR(1,4,2,8)	64	2	64	566	14	2	16
SR(1,2,4,8)	64	4	64	347	14	3	17
SR(1,4,4,8)	128	8	128	598	39	—	—

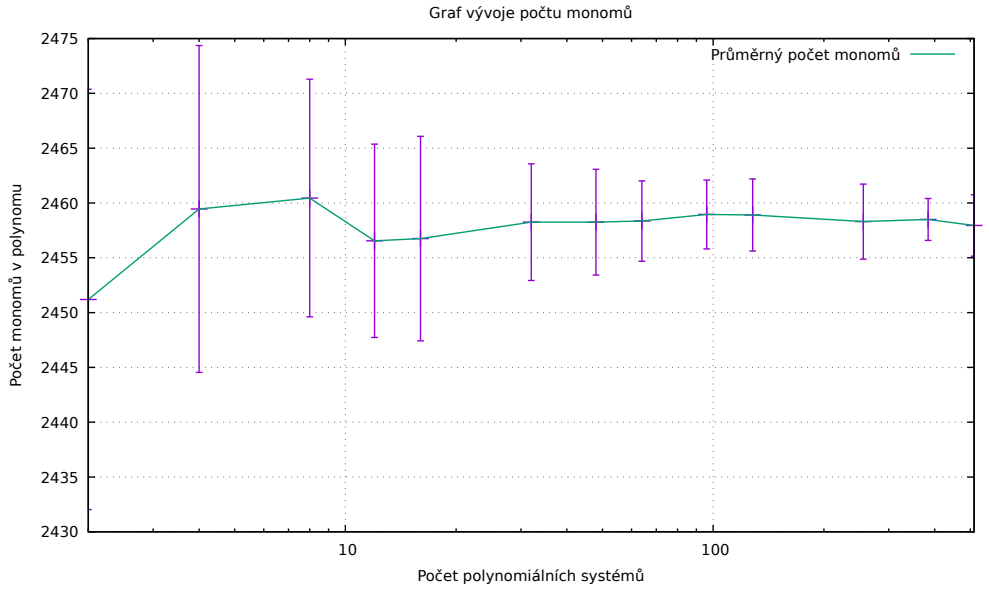
*Legenda názvů tabulky je uvedena v (4.1).

Z tabulky 4.1 vyplývá, že čím je vyšší počet rund, tím více obsahují jednotlivé polynomy monomů. To ovšem platí pouze do 3. rundy, protože od 4. se počet monomů již tolik nezvyšuje, jak je psáno v [17]. Proto jsme měření pro více rund už neprováděli.

Pro ukázku zde ještě uvedeme pár grafů, ze kterých výsledky v tabulce 4.1 vycházejí. Vybrali jsme grafy pro šifru SR(2,2,2,4) a na obrázku 4.1 tedy vidíme vývoj času přípravy polynomů i výpočtu tajného klíče v závislosti na počtu vygenerovaných polynomiálních systémů. Na obrázku 4.2 je pak vidět vývoj průměrného počtu monomů pro dané polynomy.



Obrázek 4.1: Graf vývoje průměrného času výpočtu pro různá *num* pro SR(2,2,2,4) při referenčním řešení



Obrázek 4.2: Graf vývoje průměrného počtu monomů v polynomech pro různá *num* pro SR(2,2,2,4) při referenčním řešení

4.2 PAM

Jako první metodu redukce jsme zvolili klastrování pomocí algoritmu PAM. Jako vstupní data jsme použili otevřené texty a tajné klíče vygenerované při měření referenčního řešení, abychom mohli všechny metody navzájem porovnávat.

Pro tuto metodu jsme provedli několik sad experimentů. Při každé z nich jsme měnili parametr *num* a zároveň jsme v každé sadě vybírali jiný počet polynomů pro výpočet Gröbnerovy báze. Při prvním pokusu to byl stejný počet polynomů jako bitů tajného klíče. Poté jsme počet vybraných polynomů pokaždé zdvojnásobili, dokud jsme jich neměli 8krát počet bitů klíče. Proto jsme začali naše měření až od *num*= 8.

V tabulce 4.2 uvádíme tedy tentokrát nejlepší výsledky ze všech sad experimentů provedených pro tuto metodu redukce.

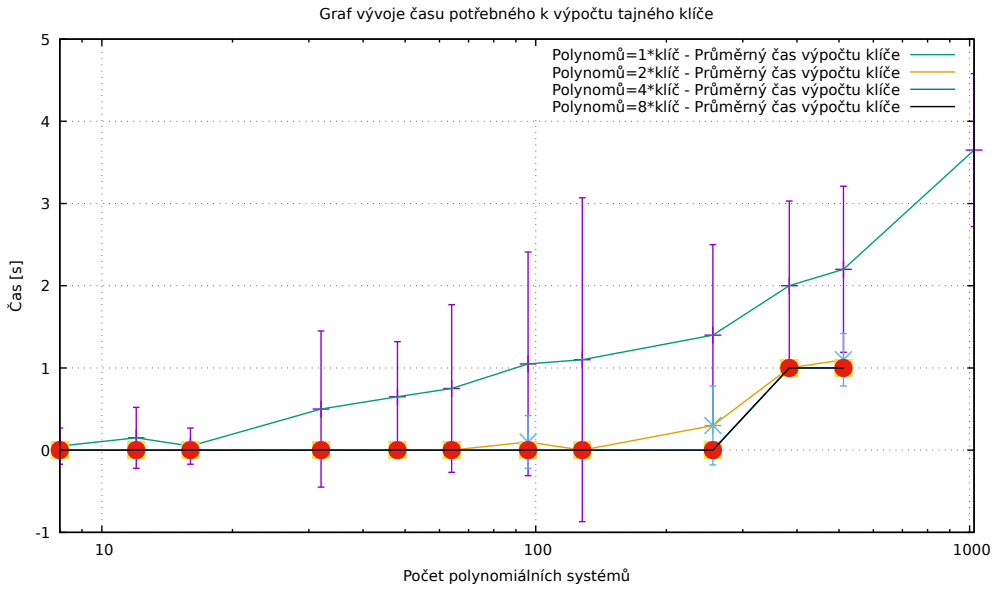
Tabulka 4.2: Naměřené časy běhu pro nejlepší konfigurace při použití metody redukce PAM

Šifra	KB [*bity*]	PS	VP	MON	ČPP [*s*]	ČVK [*s*]	ČVC [*s*]
SR(1,2,2,4)	16	8	128	19	1	< 1	1
SR(2,2,2,4)	16	8	64	1847	12	2	14
SR(3,2,2,4)	16	2	16	32598	24	647	671
SR(1,4,2,4)	32	8	256	14	2	< 1	2
SR(2,4,2,4)	32	2	32	29207	58	—	—
SR(1,2,4,4)	32	8	256	23	2	< 1	2
SR(2,2,4,4)	32	48	32	1148	5018	26	5044
SR(1,4,4,4)	64	8	512	40	11	< 1	11
SR(1,2,2,8)	32	8	256	315	17	< 1	17
SR(1,4,2,8)	64	8	512	566	99	4	103
SR(1,2,4,8)	64	8	512	364	58	46	104
SR(1,4,4,8)	128	8	256	412	438	7	445

*Legenda názvů tabulky je uvedena v (4.1).

Z celkového pohledu na všechny výsledné grafy vyplynulo, že při sadě experimentů, kde je pro výpočet vybrán stejný počet polynomů, jako je bitů klíče, trvá výpočet nejdéle. Ostatní sady mají výsledky výrazně lepší

než tato, ale navzájem jsou si poměrně podobné. Tento poměr je ukázán na obrázku 4.3.



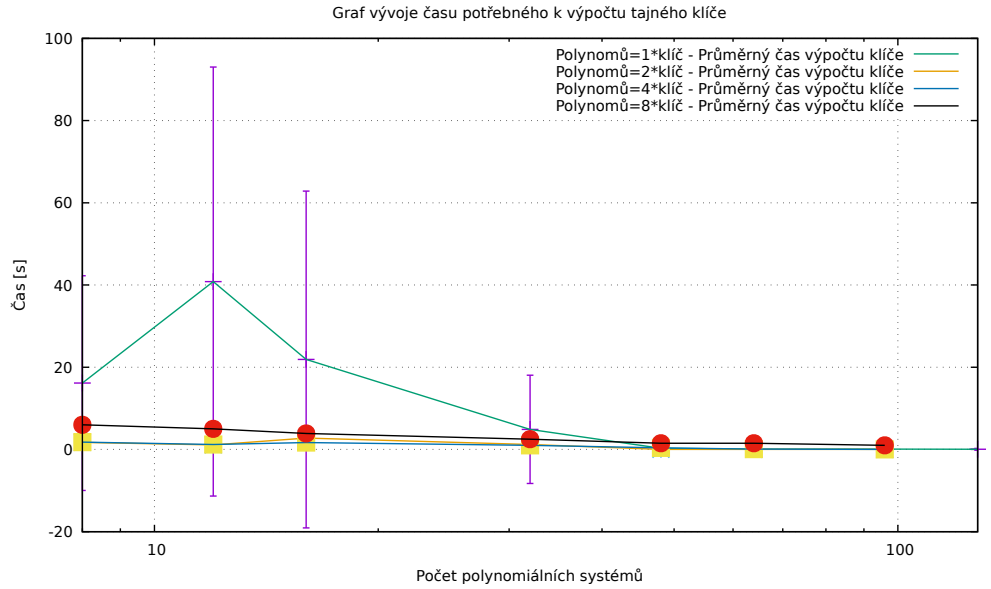
Obrázek 4.3: Graf vývoje průměrného času výpočtu klíče pro různá *num* pro SR(1,2,2,4) při metodě PAM

Zároveň z těchto grafů také vyplynulo, že čím více polynomů jsme vybrali pro výpočet, tím větší byl průměrný počet monomů na 1 polynom. Počet polynomů odpovídá v této metodě počtu klastrů, které vytváříme. Čím více tedy chceme výsledných polynomů, tím více máme klastrů.

Pro šifry SR(2,2,2,4) a SR(3,2,2,4) je překvapivé, že nejlepší výsledky nedostáváme při variantě, kdy máme nejvíce polynomů pro výpočet Gröbnerovy báze, jako je tomu u ostatních šifer. U varianty SR(2,2,2,4) jsou nejlepší výsledky pokud vybereme 4krát počet bitů klíče a pro variantu SR(3,2,2,4) nám pak dokonce stačí pouze stejný počet polynomů, jako má klíč bitů.

Varianta SR(2,2,2,4) navíc jako jediná vykazuje sestupnou závislost času na výpočet tajného klíče pro zvedající se parametr *num*. Proto si na obrázku 4.4 uvedeme graf této závislosti.

Šifru SR(2,2,4,4) se nám ze všech pokusů podařilo vypočítat pouze jednou, takže pro ni nejsou hotové žádné grafy, ale výsledný výpočet přesto v tabulce 4.2 uvádíme.



Obrázek 4.4: Graf vývoje průměrného času výpočtu klíče pro různá *num* pro SR(2,2,2,4) při metodě PAM

Pomocí této metody se nám podařilo najít tajný klíč i pro první rundu nezjednodušené šifry AES, tedy SR(1,4,4,8). Redukce však v tomto případě trvá opravdu dlouho, a tak do stanoveného limitu (3 hodiny) doběhl pouze výpočet pro maximálně 16 vygenerovaných polynomiálních systémů (*num*= 16). V tomto případě trval výpočet klíče pouze 4 vteřiny. Přestože pro variantu *num*= 8, trval výpočet klíče o 3 vteřiny déle, redukce byla rychlejší o půl hodiny, a proto jsme za nejlepší výsledek prohlásili tuto konfiguraci.

4.3 LSH

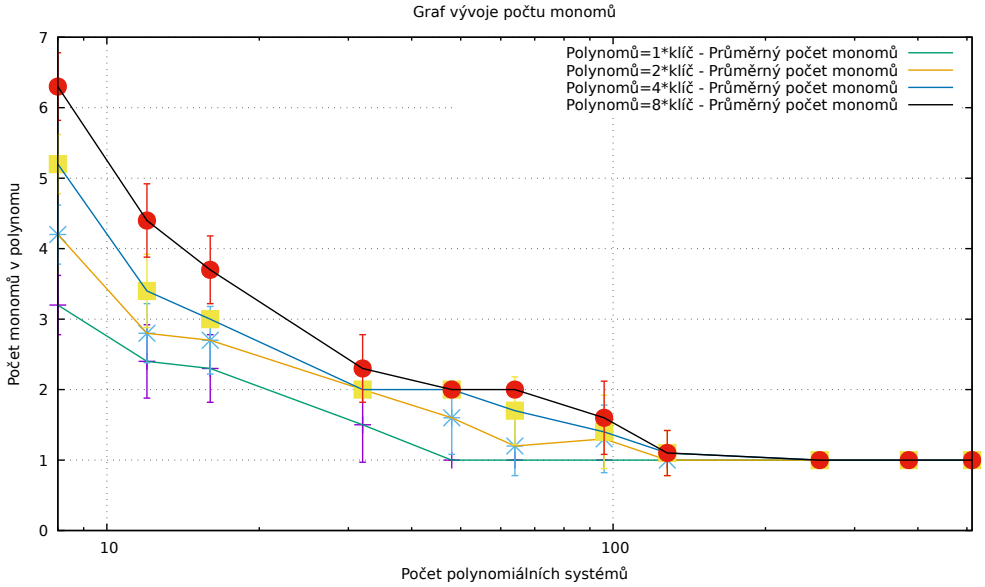
Dále jsme začali měřit metodu redukce využívající algoritmus LSH. Opět jsme zde použili již připravená vstupní data a opět jsme provedli celkem 4 sady experimentů lišící se počtem polynomů použitých pro výpočet Gröbnerovy báze. Tento počet jsme postupně nastavili na 1krát počet bitů klíče, poté 2krát, 4krát a nakonec 8krát.

V tabulce 4.3 opět uvádíme nejlepší výsledky ze všech sad experimentů provedených pro tuto metodu redukce. Pro většinu šifer platilo pravidlo, že čím více polynomů bylo vybráno pro výpočet Gröbnerovy báze, tím rychleji se tato báze spočítala. Takže pro většinu šifer jsme vybírali nejlepší výsledek ze sady, kde počet vybraných polynomů odpovídal osminásobku bitů klíče.

Tabulka 4.3: Naměřené časy běhu pro nejlepší konfigurace při použití metody redukce LSH

Šifra	KB $[bity]$	PS	VP	MON	ČPP $[s]$	ČVK $[s]$	ČVC $[s]$
SR(1,2,2,4)	16	8	128	6	1	< 1	1
SR(2,2,2,4)	16	12	128	555	3	< 1	3
SR(3,2,2,4)	16	16	16	32335	174	589	763
SR(1,4,2,4)	32	8	256	14	2	< 1	2
SR(2,4,2,4)	32	48	64	9869	1488	—	—
SR(1,2,4,4)	32	8	256	6	2	< 1	2
SR(2,2,4,4)	32	8	64	2056	24	8	32
SR(1,4,4,4)	64	8	512	13	5	< 1	5
SR(1,2,2,8)	32	8	256	200	9	< 1	9
SR(1,4,2,8)	64	8	512	405	20	14	34
SR(1,2,4,8)	64	8	512	195	17	1	18
SR(1,4,4,8)	128	48	512	288	144	3	147

*Legenda názvů tabulky je uvedena v (4.1).

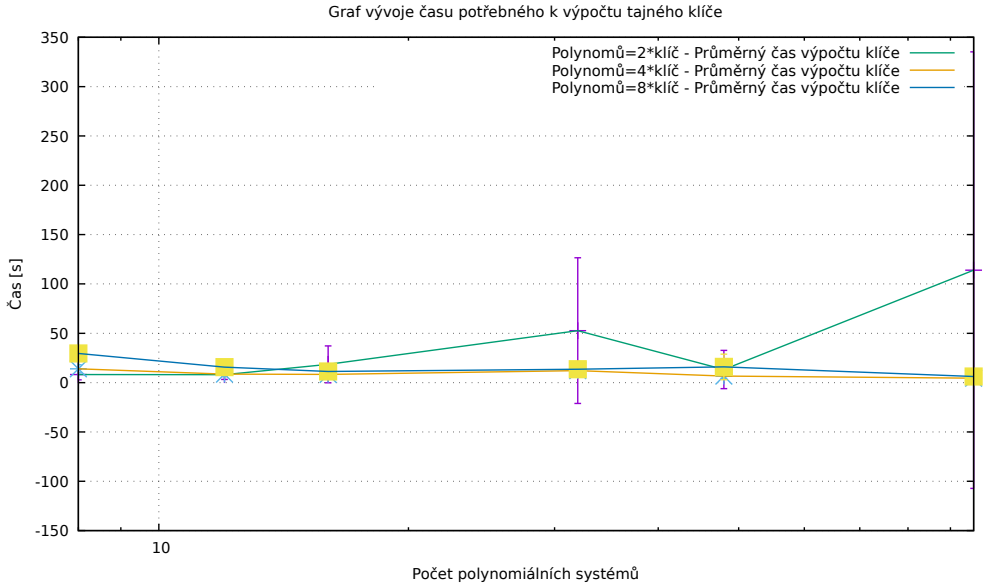


Obrázek 4.5: Graf vývoje průměrného počtu monomů v polynomech pro různá *num* pro SR(1,2,4,4) při metodě LSH

Pro jednodušší šifry dokázala tato metoda najít tak podobné polynomy, že výsledné xory obsahovaly při vyšším *num* i méně než 3 monomy, jak je vidět na obrázku 4.5. Při takto krátkých polynomech je však výpočet Gröbnerovy báze a tajného klíče naopak o něco složitější. Proto je třeba najít optimální délku polynomů tak, aby výpočet byl co nejefektivnější.

Pro šifru SR(3,2,2,4) výpočet při experimentech s více polynomy pro výpočet GB vůbec nedoběhl, a tak zde uvádíme pouze výsledek, kde se vezme stejně polynomů jako bitů klíče.

Šifra SR(2,2,4,4) je jedna z mála, u které se porušilo pravidlo, že čím více polynomů pro výpočet Gröbnerovy báze, tím lépe. To ovšem platí pouze pro velice nízký parametr *num*. Na obrázku 4.6 si tedy uvedeme vývoj času potřebného k výpočtu tajného klíče z již zredukovaných polynomů.



Obrázek 4.6: Graf vývoje průměrného času výpočtu klíče pro různá *num* pro SR(2,2,4,4) při metodě LSH

Tato metoda, stejně jako PAM, dokázala nalézt řešení i pro alespoň první rundu nezjednodušené šifry AES. Ze všech sad experimentů se tajný klíč vypočítal nejrychleji, když se pro výpočet GB vybralo $4 \cdot 128$ polynomů.

Z celkového pohledu na všechny výsledné grafy vyplynulo, že pro menší počet vygenerovaných polynomů (přibližně do 100) není velký rozdíl mezi jednotlivými sadami experimentů lišícími se počtem polynomů vybraných pro výpočet GB. Čím více polynomů jsme však generovali, tím větší byl

mezi těmito sadami rozdíl s tím, že lepší výsledky jsme dostali, pokud jsme polynomů pro výpočet vybrali více.

Zároveň z těchto grafů opět vyplynulo, stejně jako u metody PAM, že čím více polynomů jsme vybrali pro výpočet, tím větší byl průměrný počet monomů na 1 polynom. Při této metodě je vysvětlení jednoduché. Protože vybíráme polynomy z pole seřazeného podle velikosti vzdálenosti daných párů, čím více polynomů vybereme, tím jsou páry, které xorujeme, od sebe vzdálenější a výsledek xoru má tedy více monomů. Na obrázku 4.5 je demonstrována i tato vlastnost metody LSH.

4.4 Odstranění nejvýznamnějšího monomu

Poslední metodou, pro kterou bylo potřeba provést daná měření, byla metoda odstranění nejvýznamnějšího monomu. Přestože naše experimenty běžely na školním serveru prakticky bez přestání několik měsíců, bohužel jsme pro tuto metodu nestihli naměřit varianty s více vybranými polynomy pro výpočet Gröbnerovy báze.

Proto zde v tabulce 4.4 uvádíme výsledky pouze varianty, kde se vybralo přesně tolik polynomů, kolik je bitů tajného klíče pro každou šifru.

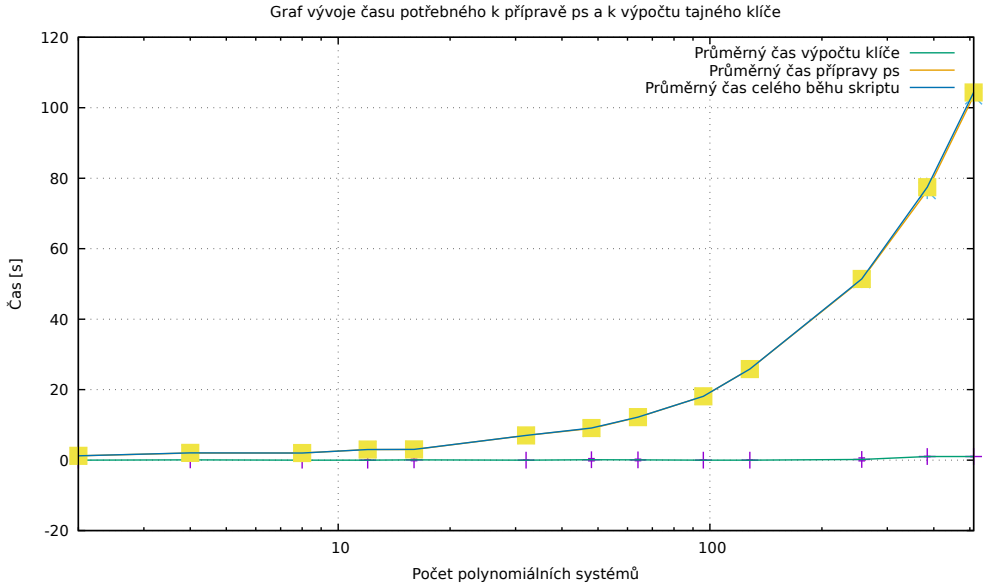
Tabulka 4.4: Naměřené časy běhu pro nejlepší konfigurace při použití metody redukce odstranění nejvýznamnějšího monomu

Šifra	KB [*bity*]	PS	VP	MON	ČPP [*s*]	ČVK [*s*]	ČVC [*s*]
SR(1,2,2,4)	16	2	16	22	< 1	< 1	1
SR(2,2,2,4)	16	4	16	771	2	6	8
SR(3,2,2,4)	16	2	16	32759	8	756	764
SR(1,4,2,4)	32	2	32	35	1	< 1	2
SR(2,4,2,4)	32	96	32	32133	185	—	—
SR(1,2,4,4)	32	2	32	26	1	< 1	2
SR(2,2,4,4)	32	8	32	3305	7	645	652
SR(1,4,4,4)	64	8	64	41	5	345	350
SR(1,2,2,8)	32	8	32	337	9	3134	3143
SR(1,4,2,8)	64	8	64	579	< 1	—	—
SR(1,2,4,8)	64	8	64	390	< 1	—	—
SR(1,4,4,8)	128	96	256	634	11	—	—

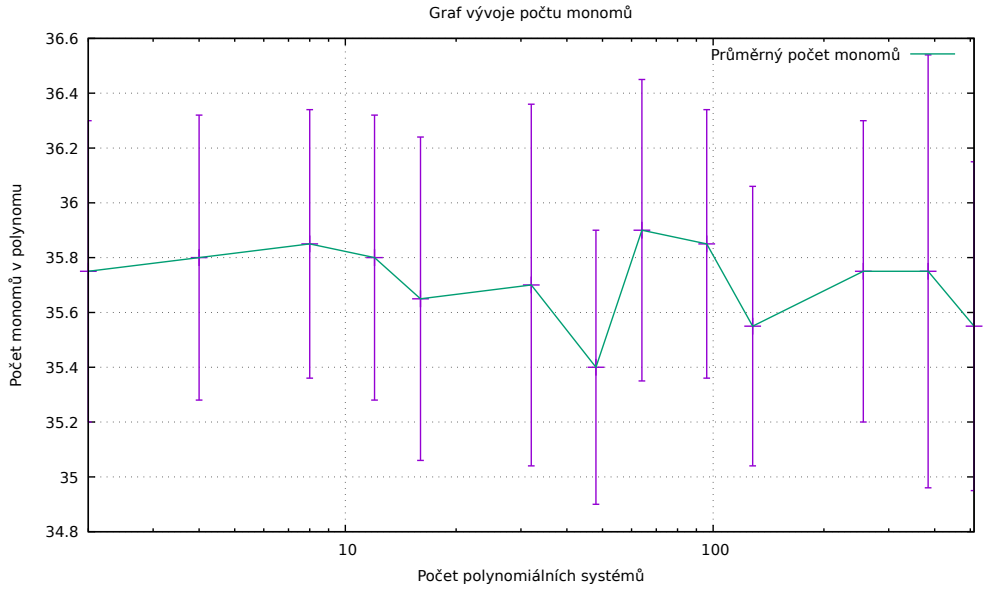
*Legenda názvů tabulky je uvedena v (4.1).

Pro SR(1,2,2,4) při zvyšujícím se počtu polynomů nenastává žádná změna. Příprava polynomů trvá až do *num*= 512 pod minutu a klíč se vždy najde do 1 vteřiny. Pro SR(2,2,2,4) pak nastala mezi 2 a 4 systémy velká změna v průměrném počtu monomů na polynom. Zde se jejich počet, přibližně 2000, zredukoval na něco málo pod 800. Tento průměrný počet pak už zůstává pro všechny ostatní počty systémů.

Pro SR(1,4,2,4) nastává přibližně od 300 systémů velký nárůst času pro redukci polynomů, ale výpočet klíče se stále vejde do 1 vteřiny. Tuto šifru si můžeme vzít jako modelovou a uvést na obrázcích 4.7 a 4.8 vývoj času přípravy polynomů i výpočtu tajného klíče a vývoj průměrného počtu monomů pro dané polynomy.



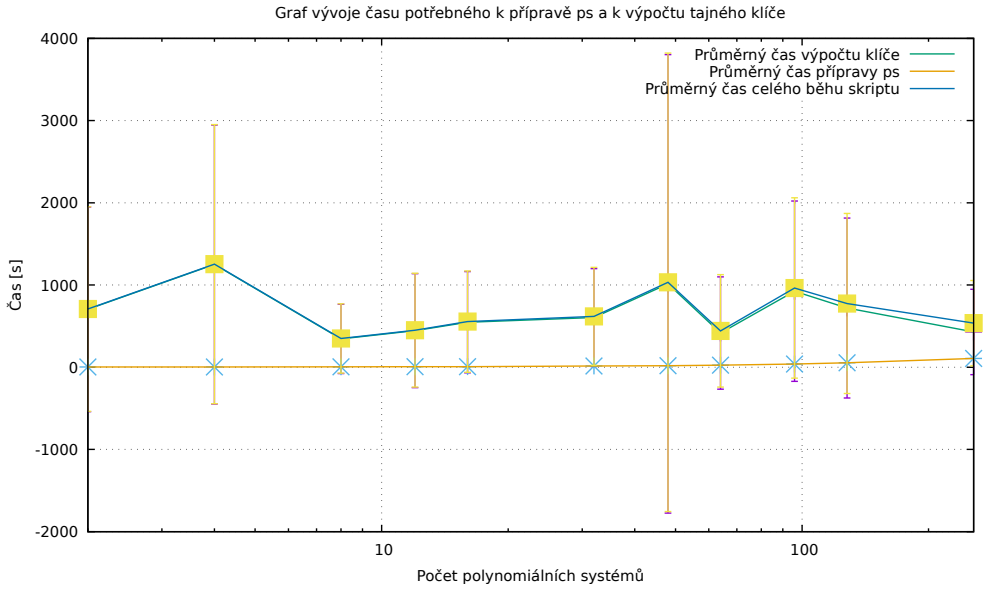
Obrázek 4.7: Graf vývoje průměrného času výpočtu pro různá *num* pro SR(1,4,2,4) při metodě odstranění nejvýznamnějšího monomu



Obrázek 4.8: Graf vývoje průměrného počtu monomů v polynomech pro různá *num* pro SR(1,4,2,4) při metodě odstranění nejvýznamnějšího monomu

Šifry SR(2,2,4,4) a SR(1,4,4,4) vykazují nejmenší míru korelace mezi počtem systémů polynomů a výsledným časem běhu algoritmu. Přestože čas redukce polynomů zde s přibývajícím počtem polynomů samozřejmě roste, tak výpočet klíče je v těchto případech tak významnou složkou celého běhu, že tuto korelaci převáží, a tak např. na obrázku 4.9 křivka více fluktuuje.

Pokud bychom doměřili i varianty, kdy se pro výpočet Gröbnerovy báze vybere více polynomů, tak na základě výsledků z předchozích metod věříme, že i pro tuto metodu bychom mohli dostat lepší výsledky a možná bychom zvládli dopočítat i některé šifry, které jsme zatím nevyřešili.



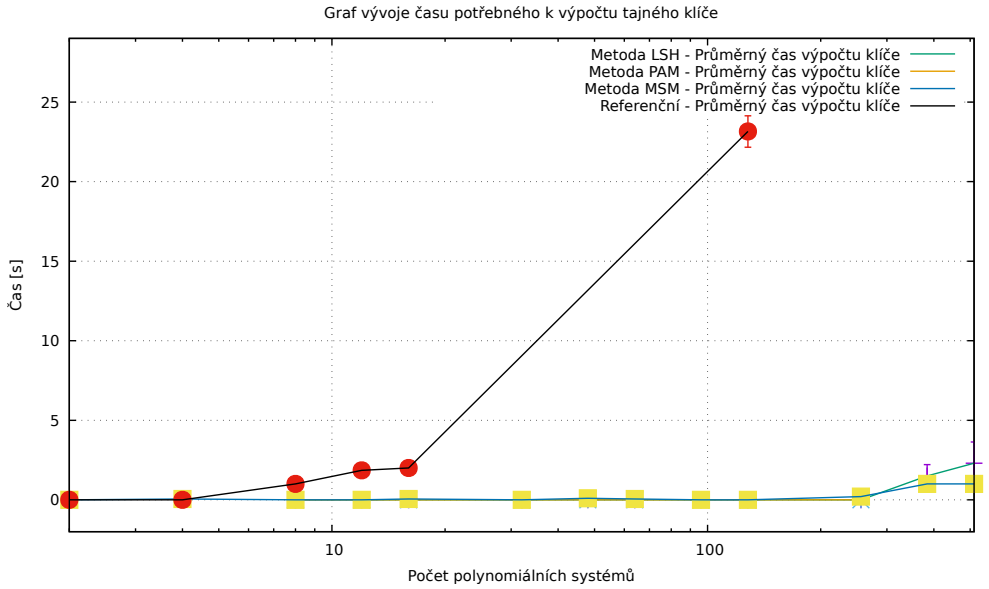
Obrázek 4.9: Graf vývoje průměrného času výpočtu pro různá *num* pro SR(1,4,4,4) při metodě odstranění nejvýznamnějšího monomu

4.5 Porovnání metod

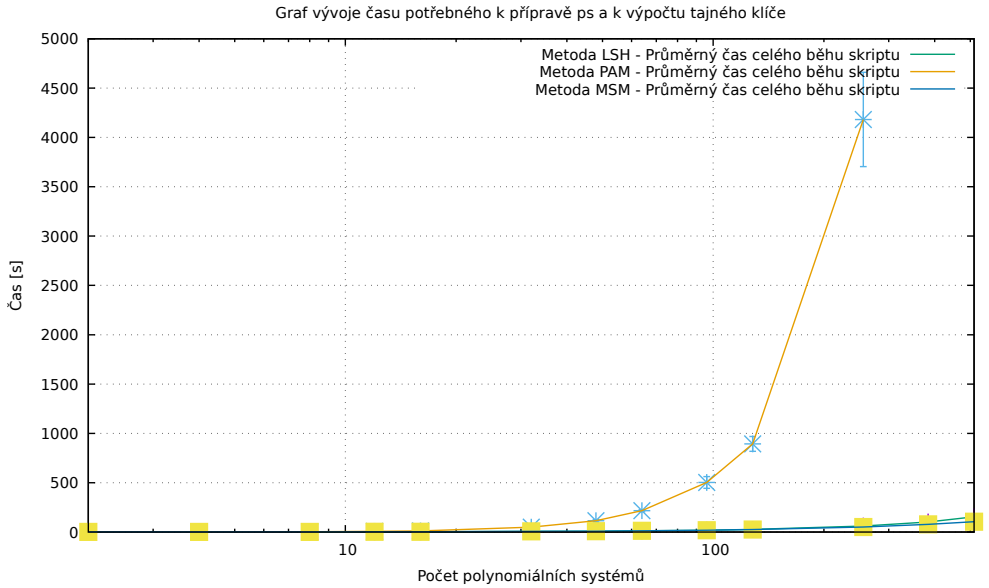
Při porovnávání všech vyzkoušených metod redukce skupiny polynomů, se můžeme na věc dívat z několika hledisek. Nejprve budeme brát v úvahu pouze čas výpočtu tajného klíče a samotná příprava polynomů nás nebude zajímat.

Oproti referenčnímu řešení si zde polepšily všechny metody. Při menším počtu polynomů k redukci není tento rozdíl tak markantní, ale když tento počet navýšíme, tak po redukci běží výpočet mnohem rychleji. Tento rozdíl je dobře vidět na obrázku 4.10.

Pokud bychom porovnávali jednotlivé metody pouze mezi sebou, tak ze všech grafů vyplývá, že výpočet tajného klíče z již zredukovaných polynomů trvá pro metody PAM a LSH téměř stejnou velice krátkou dobu. Pro metodu odstranění nejvyššího monomu byl čas výpočtu delší. Můžeme však předpokládat že za to může fakt, že pro tuto metodu jsme nestihli naměřit experimenty s větším výběrem polynomů pro výpočet klíče.



Obrázek 4.10: Graf vývoje průměrného času výpočtu klíče pro různá *num* pro SR(1,4,2,4) pro všechny metody

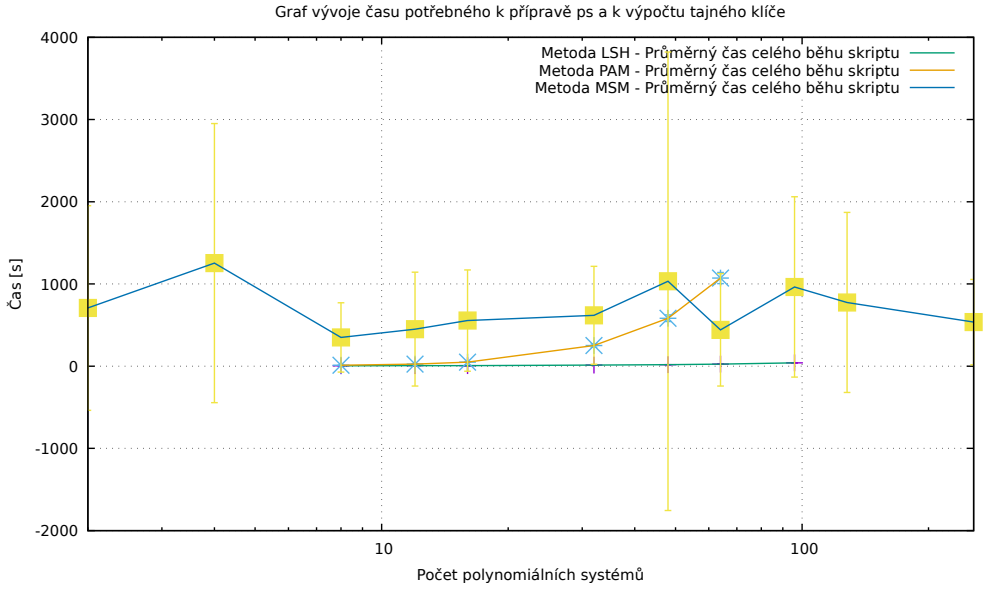


Obrázek 4.11: Graf vývoje průměrného času celého výpočtu pro různá *num* pro SR(1,4,2,4) pro všechny metody

Nyní se zaměříme na čas běhu celého skriptu, takže nás bude zajímat jak výpočet tajného klíče, tak i čas strávený generováním a redukováním všech polynomů. Na první pohled je jasné, že redukce polynomů trvala při jejich větším množství nejdéle při klastrování pomocí algoritmu PAM. Naopak, při použití algoritmu pro hledání nejbližšího souseda pomocí LSH trvala redukce i pro velké množství polynomů poměrně krátce. Na obrázku 4.11 je tento rozdíl dobře patrný.

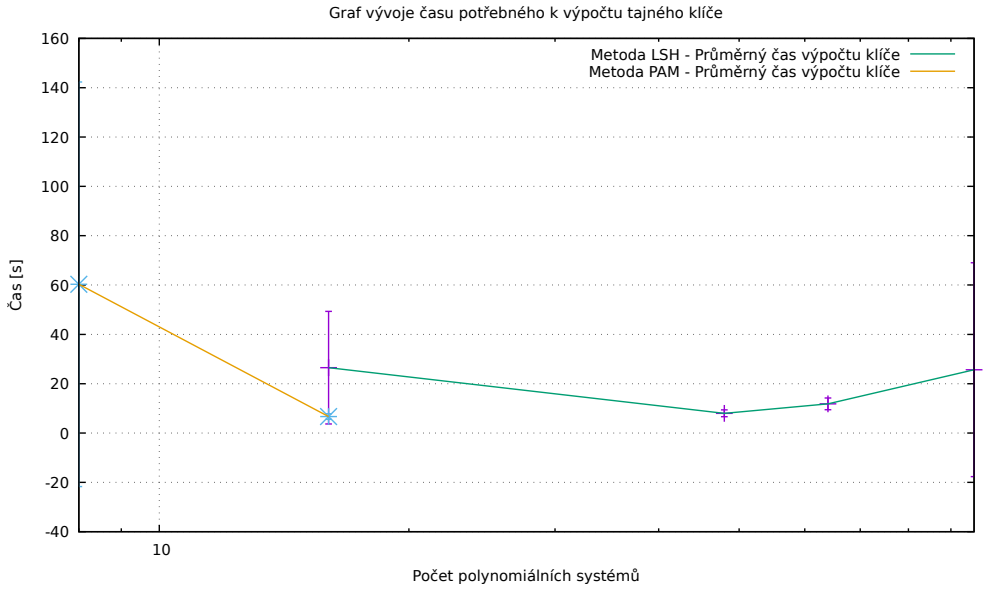
Pro metodu odstranění nejvýznamnějšího monomu už je však situace složitější. Na obrázku 4.11 je sice vidět, že oproti metodě PAM pro větší množství polynomů funguje tato metoda poměrně rychle. Když se však zaměříme pouze na malé množství polynomů, jako je tomu například na obrázku 4.12, tak vidíme, že metoda PAM je v tomto případě stále efektivnější.

Když vezmeme v úvahu oba grafy na obrázcích 4.11 a 4.12, tak vidíme, že čas výpočtu, na rozdíl od metody PAM, u metody odstranění nejvýznamnějšího monomu nijak závratně ani neroste ani neklesá (ale pouze lehce kolísá).

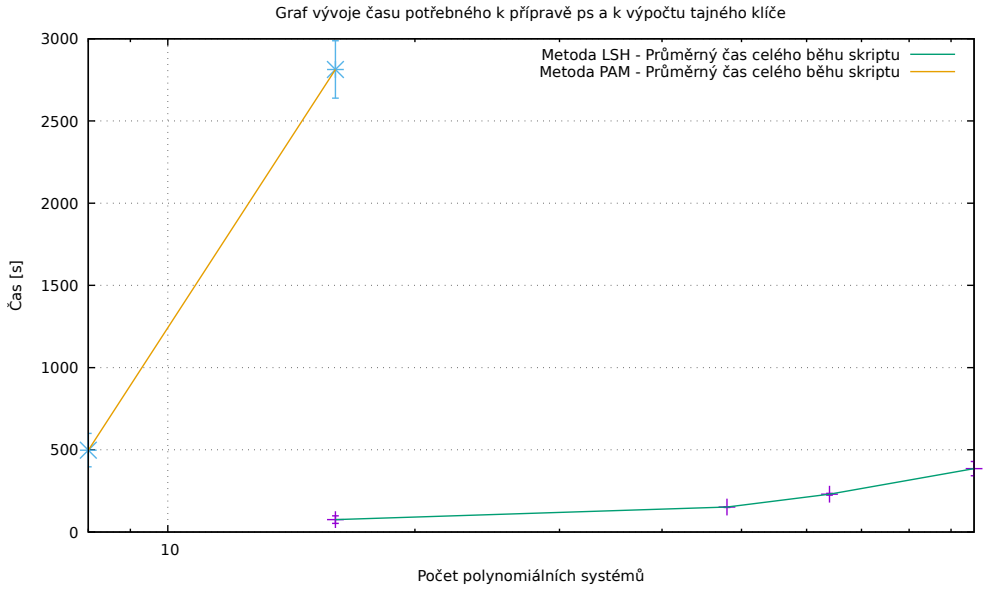


Obrázek 4.12: Graf vývoje průměrného času celého výpočtu pro různá *num* pro SR(1,4,4,4) pro všechny metody

V dalším porovnání se budeme soustředit na výsledky konkrétní šifry SR(1,4,4,8), tedy na nezjednodušenou variantu AES. Povedlo se nám prozatím prolomit pouze její první rundu pomocí 2 metod - PAM a LSH.



Obrázek 4.13: Graf vývoje průměrného času výpočtu klíče pro různá *num* pro SR(1,4,4,8) pro PAM a LSH



Obrázek 4.14: Graf vývoje průměrného času celého výpočtu pro různá *num* pro SR(1,4,4,8) pro PAM a LSH

Metoda PAM dokázala tajný klíč vypočítat pro stejný parametr *num* o trochu rychleji než LSH, což je vidět na obrázku 4.13. Na druhou stranu, redukce při použití klastrování trvala řádově delší dobu než u LSH i pro malý počet vygenerovaných polynomů, takže výsledný čas celého výpočtu byl pro metodu PAM mnohem delší. To ukazuje obrázek 4.14. Proto můžeme metodu LSH prohlásit za efektivnější.

Nakonec ještě můžeme porovnat naše výsledky s těmi, ke kterým dospěl ve své diplomové práci Ing. Bielik [13]. Pro většinu šifer odhalil tajný klíč po 1. rundě v řádu jednotek vteřin za použití 2 polynomiálních systémů bez redukce. Tyto výsledky odpovídají našemu referenčnímu řešení, které jsme pak našimi metodami redukce ještě vylepšili.

Pro 2. rundy šifer pak pan Bielik použil redukci, jež k polynomům v 1 základním systému hledá nejpodobnější v množině polynomů z několika málo dalších systémů a ty pak vzájemně xoroje. Díky ní, se mu za 1 hodinu podařilo vypočítat tajný klíč šifry SR(2,2,4,4).

Při použití naší metody PAM se klíč vypočítal za 26 vteřin, ovšem redukce polynomů trvala neúměrně dlouho. Při metodě LHS však už redukce běžela mnohem rychleji a celý výpočet trval 32 vteřin, z toho pouze 8 vteřin bylo potřeba na odhalení klíče.

Šifru SR(2,4,2,4) ani žádné další náročnější varianty se však nepodařilo vyřešit při žádné redukci nám ani panu Bileikovi.

Jedinou šifrou, pro kterou se nám oběma podařilo vyřešit klíč po 3. rundě byla SR(3,2,2,4). Po redukci pana Bielika trval výpočet klíče přes čtvrt hodiny, kdežto při použití našich metod PAM a LSH to bylo průměrně pouze 10 minut při odstranění nejvýznamnějšího monomu 12,6 minut.

Pan Bielik pak svůj výsledek zlepšil ještě použitím navíc útoku *guess-and-determine* a vypočítal klíč za pouhých 6 vteřin. Navíc se mu podařilo za necelou minutu vyřešit i šifru SR(2,4,2,4). Pokud bychom toto vylepšení zkombinovali s našimi metodami, věříme že bychom dosáhli opravdu výborných výsledků.

Závěr

Naším prvním úkolem bylo popsat Gröbnerovy báze a algoritmy pro jejich výpočet a seznámit čtenáře se šifrou AES a s možnostmi její kryptoanalýzy. V 1. kapitole jsme tedy zavedli všechny nezbytné matematické pojmy a postupně se propracovali až k Buchbergerovu algoritmu a algoritmu F4.

Ve 2. kapitole jsme se pak zaměřili na šifru AES. Stručně jsme popsali, jakým způsobem funguje celý algoritmus šifrování a představili jsme si její zjednodušené varianty. Dále jsme vysvětlili, jak se dají tyto šifry převést na systém polynomiálních rovnic a představili jsme si skripty Ing. Bielika, které umí tyto rovnice vygenerovat a vyřešit.

Naším dalším úkolem pak bylo tyto skripty zanalyzovat a rozšířit o různé metody redukce předefinovaného systému. Ve 3. kapitole jsme tedy popsali implementaci redukce pomocí klastrování algoritmem PAM, nalezení nejbližšího souseda pro vzdálenost založenou na operaci xor pomocí algoritmu ze skupiny LSH a jako poslední metodu jsme si představili algoritmus pro odstranění nejvýznamnějšího monomu z nejkratších polynomů.

Hlavním cílem této práce pak bylo provést řadu experimentů, při nichž jsme měřili čas běhu jednotlivých částí našich skriptů pro všechny výše uvedené metody a snažili se najít nejlepší konfiguraci tak, aby byl výpočet tajného klíče co nejefektivnější. Vytvořili jsme si tedy řadu dalších skriptů, které spouštěly jednotlivá měření na školním serveru.

Tato měření pak běžela bez přestání po několik měsíců, abychom měli co nejvíce dat, ze kterých jsme pak vycházeli při prezentaci výsledků. Prozatím jsme se zaměřili pouze na jednodušší varianty šifry AES.

Pro všechny složitější varianty šifry AES trvá už jen vygenerování množství polynomů relativně dlouho, a proto jsme z časových důvodů tyto experimenty nemohli provést před odevzdáním této práce. Stále je ale máme na paměti a další experimenty ještě budeme provádět.

Ve 4. kapitole této práce jsme pak vypsali prozatímní nejlepší výsledky pro každou metodu redukce polynomů a popsali jejich nejdůležitější vlastnosti. Nakonec jsme pak jednotlivé metody porovnali mezi sebou a dospěli jsme k závěru, že nejvýhodnější je použití metody xorování nejbližších sousedů nalezených pomocí algoritmu LSH.

Vzhledem k tomu, že momentálně jsou všechny redukce implementovány sekvenčně, v budoucnu by možným vylepšením této práce mohla být například jejich paralelizace.

Literatura

[1] Bielik, M.: Yet Another Algebraic Cryptanalysis of Small Scale Variants of AES. [online], prosinec 2020, [cit. 2022-04-23]. Dostupné z: https://gitlab.com/upbqdn/yaac/-/tree/main/experiments

[2] Cox, D. A.; Little, J.; O'Shea, D.: *Ideals, Varieties, and Algorithms: An Introduction to Computational Algebraic Geometry and Commutative Algebra*. Cham: Springer, čtvrté vydání, 2015, ISBN 978-3-319-16721-3, doi:10.1007/978-3-319-16721-3.

[3] Buchberger, B.: Bruno Buchberger's PhD thesis 1965: An algorithm for finding the basis elements of the residue class ring of a zero dimensional polynomial ideal. *Journal of Symbolic Computation*, ročník 41, č. 3, 2006: s. 475–511, doi:10.1016/j.jsc.2005.09.007. Dostupné z: https://www.sciencedirect.com/science/article/pii/S0747717105001483

[4] Faugére, J.-C.: A new efficient algorithm for computing Gröbner bases (F4). *Journal of Pure and Applied Algebra*, ročník 139, č. 1, 1999: s. 61–88, doi:doi.org/10.1016/S0022-4049(99)00005-5.

[5] Giovini, A.; Mora, T.; Niesi, G.; aj.: "One Sugar cube, Please" or Selection Strategies in the Buchberger Algorithm. In *ISSAC 91: International Symposium on Symbolic Algebraic Computation Bonn West Germany*, New York: Association for Computing Machinery, červen 1991, s. 49–54, doi:10.1145/120694.120701.

[6] Soukup, J.: *Algebraická kryptoanalýza proudových šifer založených na LFSR*. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, květen 2022, diplomová práce.

[7] Nair, M. T.; Singh, A.: *Linear Algebra*. Singapore: Springer, 2018, ISBN 978-981-13-0926-7, doi:10.1007/978-981-13-0926-7. Dostupné z: https://link.springer.com/book/10.1007/978-981-13-0926-7

[8] NIST FIPS, P.: 197: Advanced encryption standard (AES). *Federal information processing standards publication*, ročník 197, listopad 2001.

[9] Daemen, J.; Rijmen, V.: *The design of Rijndael*, ročník 2. Berlin, Heidelberg: Springer, 2002, ISBN 978-3-662-04722-4, doi:10.1007/978-3-662-04722-4.

[10] Cid, C.; Murphy, S.; Robshaw, M. J. B.: Small Scale Variants of the AES. In *Fast Software Encryption*, editace H. Gilbert; H. Handschuh, Berlin, Heidelberg: Springer, 2005, ISBN 978-3-540-31669-5, s. 145–162, doi:10.1007/11502760_10.

[11] Cid, C.; Murphy, S.; Robshaw, M.: *Algebraic Aspects of the Advanced Encryption Standard*. New York: Springer, 2006, ISBN 978-0-387-36842-9, doi:10.1007/978-0-387-36842-9. Dostupné z: hhttps://link.springer.com/book/10.1007/978-0-387-36842-9

[12] Bulygin, S.; Brickenstein, M.: Obtaining and solving systems of equations in key variables only for the small variants of AES. *Mathematics in Computer Science*, ročník 3, č. 2, 2010: s. 185–200, doi:10.1007/s11786-009-0020-y. Dostupné z: https://link.springer.com/content/pdf/10.1007/s11786-009-0020-y.pdf

[13] Bielik, M.: *Algebraic Cryptanalysis of Small Scale Variants of the AES*. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, leden 2021, diplomová práce.

[14] Bosma, W.; Cannon, J.; Playoust, C.: The Magma Algebra System I: The User Language. *Journal of Symbolic Computation*, ročník 24, č. 3, 1997: s. 235–265, doi:doi.org/10.1006/jsco.1996.0125.

[15] viz. soubor AUTHORS: CryptoMiniSat SAT solver. [online], 2009-2020, [cit. 2022-04-20]. Dostupné z: https://github.com/msoos/cryptominisat

[16] Brickenstein, M.; Dreyer, A.: PolyBoRi: A framework for Gröbner-basis computations with Boolean polynomials. *Journal of Symbolic Computation*, ročník 44, č. 9, 2009: s. 1326–1345, doi:10.1016/j.jsc.2008.02.017.

[17] Bielik, M.; Jureček, M.; Jurečková, O.; aj.: Yet Another Algebraic Cryptanalysis of Small Scale Variants of AES. Cryptology ePrint Archive, Paper 2022/695, 2022. Dostupné z: https://eprint.iacr.org/2022/695

[18] Kaufman, L.; Rousseeuw, P.: *Finding Groups in Data: An Introduction To Cluster Analysis*. NewYork: John Wiley, leden 1990, ISBN 0471878766, doi:10.2307/2532178.

[19] Bryant, V.: *Metric Spaces: Iteration and Application*. Cambridge University Press, 1985, ISBN 9780521318976.

[20] Lewin, M.: All About XOR. [online], červen 2012, [cit. 2022-07-01]. Dostupné z: https://accu.org/journals/overload/20/109/lewin_1915/

[21] Wang, J.; Shen, H. T.; Song, J.; aj.: Hashing for Similarity Search: A Survey. *CoRR*, ročník abs/1408.2927, 2014, doi:10.48550/ARXIV.1408.2927, [cit. 2022-08-27]. Dostupné z: https://arxiv.org/abs/1408.2927

[22] Briggs, J.: Faiss: The Missing Manual. [online], [cit. 2022-07-27]. Dostupné z: https://www.pinecone.io/learn/locality-sensitive-hashing/

Příloha **A**

Seznam použitých zkratek

AES *Advanced Encryption Standard*, bloková šifra

$Z_{\geq 0}^{n}$ množina celých čísel větších nebo rovna nule

$\oplus$**, xor** *exclusive or*, exkluzivní disjunkce

klastrování shluková analýza

klastr shluk - skupina podobných prvků nějaké množiny

PAM *Partition Around Medoids*, algoritmus pro nehierarchickou shlukovou analýzu

mediod centrum klastru, jež je prvkem původní množiny

LSH *Locality Sensitive Hashing*, hashování zaměřené na vzdálenost

hash zkrácená podoba dat, vytvořená z nich pomocí hashovací funkce

Příloha B

Obsah přiloženého CD

- `README.txt` ..... stručný popis obsahu CD
- `DP_Beruskova_Jana_2022.pdf` ..... text práce ve formátu PDF
- `text/` ..... adresář se zdrojovou formou práce ve formátu Latex
  - `obr/` ..... adresář s obrázky použitými v textu práce
- `skripty/` ..... adresář se skripty pro výpočet tajného klíče šifry AES
  - `experiments.py` ..... skript, kterým se spouští výpočet klíče
  - `saes.py` ..... skript, který počítá tajný klíč
  - `config.py` ..... skript obsahující potřebnou konfiguraci
- `experimenty/` ..... adresář se všemi provedenými měřeními
  - `spousteni/` ... adresář se skripty pro spouštění jednotlivých měření
    - `textdata/` ... adresář se soubory s daty, které obsahují otevřené texty a klíče pro jednotlivé experimenty
    - `experiments.py` ..... skript, kterým se spouští výpočet klíče
    - `experiments_smore.py` ... upravený skript pro výpočet klíče pro více polynomů pro Magmu
    - `run_LSH_data.sh` ..... příklad měřícího skriptu pro metodu LSH
  - `vystupy/` ..... adresář s daty, vygenerovanými měřícími skripty
    - `get_data.py` ..... skript pro výtah potřebných dat pro gnuplot
    - `output_LSH_data1t.txt` ..... příklad výstupu měřícího skriptu
  - `data/` ..... adresář s daty ve vhodném formátu pro gnuplot
    - `data_lsh_1224.txt` příklad souboru s daty ve vhodném formátu pro gnuplot
    - `gnuplot_lsh1.txt` ..... příklad skriptu generujícího grafy
    - `gnuplot_cmpall.txt` .... další příklad skriptu generujícího grafy
  - `grafy/` ..... adresář s adresáři s grafy pro jednotlivé meoty
    - `ref/` ..... adresář s grafy při referenčním řešení bez ML
    - `lshall/` ..... adresář s grafy při použití algoritmu LSH
    - `pamall/` ..... adresář s grafy při použití algoritmu PAM
    - `msm1/` ... adresář s grafy při použití algoritmu nejvýznamnějšího monomu
    - `cmpall/` ..... adresář s grafy pro porovnání metod
    - `aes/` ..... adresář s grafy pro AES pro všechny metody