



## ZADÁNÍ DIPLOMOVÉ PRÁCE

**Název:** Asymetrický šifrovací algoritmus McEliece  
**Student:** Bc. Vojtěch Myslivec  
**Vedoucí:** prof. Ing. Róbert Lórencz, CSc.  
**Studijní program:** Informatika  
**Studijní obor:** Počítačová bezpečnost  
**Katedra:** Katedra počítačových systémů  
**Platnost zadání:** Do konce letního semestru 2016/17

### Pokyny pro vypracování

Prostudujte asymetrický šifrovací algoritmus McEliece založený na binárních Goppa kódech. Proveďte rešerši existujících kryptoanalýz algoritmu McEliece a jeho variant. Zvažte metody zabývající se zkrácením velikosti klíče. Implementujte šifrovací a dešifrovací algoritmy a změřte jejich výpočetní časovou a prostorovou náročnost v závislosti na velikosti klíče.

### Seznam odborné literatury

Dodá vedoucí práce.

L.S.

prof. Ing. Róbert Lórencz, CSc.  
vedoucí katedry

prof. Ing. Pavel Tvrdlík, CSc.  
děkan

V Praze dne 2. února 2016



ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE  
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ  
KATEDRA POČÍTAČOVÝCH SYSTÉMŮ



Diplomová práce

## **Asymetrický šifrovací algoritmus McEliece**

*Bc. Vojtěch Myslivec*

Vedoucí práce: prof. Ing. Róbert Lórencz, CSc.

10. května 2016



---

## Poděkování

Předně bych chtěl poděkovat Róbertu Lórenczovi za konstruktivní kritiku při vedení této práce a hlavně za velmi zajímavé téma, které mi nabídl k vypracování. Nejvíce děkuji své rodině za podporu, veškerou pomoc a hlavně trpělivost. Bez nich by tato práce nevznikla. Děkuji Tomáši Kalvodovi, za časté konzultace v oblasti matematiky a Jaroslavu Kotilovi za radu ohledně Goppa kódů. Nakonec bych rád poděkoval Ondřeji Guthovi za vytvoření použité šablony pro L<sup>A</sup>T<sub>E</sub>X.



---

## Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval(a) samostatně a že jsem uvedl(a) veškeré použité informační zdroje v souladu s Metodickým pokynem o etické 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. § 46 odst. 6 tohoto zákona 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, a za jakýmkoli účelem (včetně užití k výdělečným účelům). Toto oprávnění je časově, teritoriálně i množstevně neomezené. Každá osoba, která využije výše uvedenou licenci, se však zavazuje udělit ke každému dílu, které vznikne (byť jen zčásti) na základě Díla, úpravou Díla, spojením Díla s jiným dílem, zařazením Díla do díla souborného či zpracováním Díla (včetně překladu), licenci alespoň ve výše uvedeném rozsahu a zároveň zpřístupnit zdrojový kód takového díla alespoň srovnatelným způsobem a ve srovnatelném rozsahu, jako je zpřístupněn zdrojový kód Díla.

V Praze dne 10. května 2016

.....

České vysoké učení technické v Praze

Fakulta informačních technologií

© 2016 Vojtěch Myslivec. Všechna práva vyhrazena.

*Tato práce vznikla jako školní dílo na Českém vysokém učení technickém v Praze, Fakultě informačních technologií. Práce je chráněna právními předpisy a mezinárodními úmluvami o právu autorském a právech souvisejících s právem autorským. K jejímu užití, s výjimkou bezúplatných zákonných licencí, je nezbytný souhlas autora.*

### **Odkaz na tuto práci**

Myslivec, Vojtěch. *Asymetrický šifrovací algoritmus McEliece*. Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016. Dostupné online <https://github.com/VojtechMyslivec/mceliece-mathematica>



---

# Abstrakt

V této práci se zabýváme asymetrickým kryptosystémem *McEliece*, který je založený na samoopravných lineárních kódech a je jedním z kandidátů pro asymetrickou postkvantovou kryptografii. V práci uvádíme základní definici tohoto kryptosystému, variantu pro digitální podpis a věnujeme se též existujícím kryptoanalýzám a praktickým aspektům tohoto systému. V rámci práce vznikla ukázková implementace v softwaru *Wolfram Mathematica*, na které bylo provedeno měření časových závislostí algoritmů.

**Klíčová slova** McEliece, asymetrická kryptografie, postkvantová kryptografie, binární Goppa kódy, konečná tělesa, polynomy, Wolfram Mathematica

---

# Abstract

In this work, we deal with a code-based public-key cryptosystem *McEliece* which is one of the candidates for post-quantum cryptography. We provide a definition of the cryptosystem, its variant for digital signature scheme, and we focus on the practical aspects of this cryptosystem and its cryptanalysis. We evaluate the time complexity of the algorithms using an illustrative implementation in *Wolfram Mathematica*.

**Keywords** McEliece, public-key cryptography, post-quantum cryptography, binary Goppa codes, finite fields, polynomials, Wolfram Mathematica

---

# Obsah

<b>Úvod</b>	<b>1</b>
<b>1 Kryptosystém McEliece</b>	<b>3</b>
1.1 Generování klíčů . . . . .	4
1.2 Algoritmy pro šifrování a dešifrování . . . . .	4
1.3 Základní vlastnosti kryptosystému . . . . .	5
1.4 Bezpečnost kryptosystému . . . . .	6
<b>2 Elektronický podpis</b>	<b>9</b>
2.1 Kryptosystém Niederreiter . . . . .	9
2.2 Schéma pro elektronický podpis . . . . .	11
2.3 Algoritmy schématu pro digitální podpis . . . . .	13
<b>3 Binární Goppa kódy</b>	<b>15</b>
3.1 Sestrojení Goppa kódu . . . . .	16
3.2 Dekódování . . . . .	18
<b>4 Kryptoanalýza systému McEliece</b>	<b>21</b>
4.1 Útoky na McEliece . . . . .	21
4.2 Bezpečné parametry . . . . .	24
4.3 Slabiny kryptosystému . . . . .	26
<b>5 Moderní varianty a úpravy</b>	<b>29</b>
5.1 Metody na snížení velikosti klíčů . . . . .	29
5.2 CCA2 odolná konverze . . . . .	31
<b>6 Implementace</b>	<b>35</b>
6.1 Binární konečná tělesa . . . . .	35
6.2 Ireducibilní binární Goppa kódy . . . . .	47
6.3 McEliece . . . . .	53

<b>7</b>	<b>Analýza složitosti</b>	<b>59</b>
7.1	Velikosti klíčů . . . . .	59
7.2	Experimentální výsledky . . . . .	61
	<b>Závěr</b>	<b>69</b>
	<b>Literatura</b>	<b>71</b>
<b>A</b>	<b>Obecná algebra</b>	<b>77</b>
A.1	Základní termíny . . . . .	77
A.2	Reprezentace prvků . . . . .	78
A.3	Operace v tělese $GF(p^n)$ . . . . .	78
A.4	Rozšířená tělesa . . . . .	81
<b>B</b>	<b>Teorie kódování</b>	<b>83</b>
B.1	Samoopravné kódy . . . . .	83
B.2	Lineární kódy . . . . .	85
<b>C</b>	<b>Seznam použitých zkratk</b>	<b>91</b>
<b>D</b>	<b>Obsah příloženého disku</b>	<b>93</b>

---

## Seznam obrázků

5.1	CCA2 odolná konverze . . . . .	32
7.1	Časová složitost generování klíčů . . . . .	62
7.2	Časová složitost šifrování . . . . .	63
7.3	Časová složitost dešifrování . . . . .	63
7.4	Časová složitost generování klíčů . . . . .	64
7.5	Časová složitost šifrování . . . . .	65
7.6	Časová složitost dešifrování . . . . .	65
7.7	Poměr částí výpočtu při generování klíčů . . . . .	66
7.8	Poměr částí výpočtu při dešifrování . . . . .	66
B.1	Značení v teorii kódování . . . . .	84
B.2	Ilustrace kódové vzdálenosti . . . . .	85



---

## Seznam tabulek

4.1	Míra bezpečnosti . . . . .	25
4.2	Míra bezpečnosti . . . . .	25
4.3	Porovnání <i>McEliece</i> a <i>RSA</i> . . . . .	25
5.1	Délky vektorů v CCA2 odolné konverzi $\gamma$ . . . . .	31
5.2	Značení v CCA2 odolné konverzi $\gamma$ . . . . .	32
6.1	Syntaxe Wolfram Mathematica . . . . .	38





---

## Seznam algoritmů

1	Konverze Kobara-Imai $\gamma$ . . . . .	33
2	Sčítání prvků . . . . .	39
3	Redukce prvku . . . . .	39
4	Násobení prvků . . . . .	40
5	Rozšířený Euklidův algoritmus . . . . .	42
6	Umocňování na druhou . . . . .	43
7	Square-and-Multiply . . . . .	44
8	Generování Goppa kódu . . . . .	48
9	Dekódování Goppa kódu . . . . .	49
10	Generování klíčů pro McEliece . . . . .	53
11	Šifrování McEliece . . . . .	54
12	Dešifrování McEliece . . . . .	55



---

# Úvod

Kryptosystém *McEliece* je asymetrický šifrovací algoritmus, publikovaný poprvé v roce 1978 Robertem McEliece [1]. Je to tedy jedna z nejstarších asymetrických šifer a je první šifrou využívající teorii kódování respektive samoopravné lineární kódy.

Tento algoritmus není v praxi využíván a to hlavně kvůli velikosti klíčů, které jsou mnohonásobně větší, než klíče u aktuálně používaných šifer (*ECDSA*, *RSA*, ...). Nicméně popularita tohoto kryptosystému v poslední době roste, a to především díky tomu, že je to jeden z kandidátů pro postkvantovou asymetrickou kryptografii.

## Motivace a cíl práce

Za téměř 40 let, co je algoritmus veřejně znám, nebyl nalezený algoritmus pro kvantový počítač, který by dokázal *kryptosystém* prolomit rychleji, než na běžných počítačích [14]. Mezi kandidáty pro tzv. postkvantovou dobu je uveden např. v publikaci *Post-Quantum Cryptography* autorů *Bernstein a spol.* [8]. Dále varianta *kryptosystému* s kvazi-cyklickými *MDPC* kódy se vyskytla v draftu z roku 2016 společnosti *IETF* mezi doporučenými *postkvantovými* asymetrickými kryptosystémy [38].

V posledních letech se tomuto *kryptosystému* (jako i dalším postkvantovým algoritmům) věnuje čím dál více pozornosti. Cílem práce je získat ucelený pohled na tento *kryptosystém* a demonstrovat použití tohoto algoritmu na ukázkové implementaci. Dále je důležité prozkoumat existující kryptoanalýzy tohoto kryptosystému a zdůraznit jeho slabiny. V práci se budeme věnovat dílčí oblasti a algoritmům, které tento kryptosystém využívá. Na konci práce se věnujeme měření výpočtu a zdůrazníme tak kritická místa, na které je dobré se soustředit.

### Struktura práce

V 1. kapitole představíme *kryptosystém McEliece* tak, jak byl definován v původní publikaci *Roberta McEliece*, uvedeme jeho základní vlastnosti a ukážeme problém prolomení tohoto kryptosystému. V 2. kapitole představíme příbuzný *kryptosystém Niederreiter* a jeho použití pro získání digitálního podpisu.

V původní publikaci byly pro použití kryptosystému doporučeny tzv. *binární Goppa kódy*, které popíšeme v 3 kapitole. Pokusy o nahrazení těchto kódů jinými, kompaktnějšími kódy se často ukázaly jako nedostatečně odolné vůči prolomení soukromého klíče či šifrovaného textu. Kryptoanalýzou a případnými slabinami *kryptosystému* se věnujeme v 4 kapitole.

Aby byl *kryptosystém* využitelný v praktických aplikacích, existuje několik moderních variant, včetně konverze pro zabránění útoku s voleným šifrovým textem. Těmto moderním variantám se věnujeme v kapitole 5.

Kapitola 6 se zabývá ukázkovou implementací tohoto kryptosystému v softwaru *Wolfram Mathematica* a v poslední kapitole 7 jsou demonstrovány výsledky z provedených měření složitostí algoritmů na této implementaci.

**Poznámka:** V práci předpokládáme základní znalosti z teorie kódování a obecné algebry. Tyto znalosti jsou potřeba pro pochopení definice *kryptosystému* a též i jeho použití. Pro případné ujasnění těchto poznatků uvádíme informace věnující se obecné algebře v příloze A a teorii kódování v příloze B.

# Kryptosystém McEliece

V této kapitole popisujeme kryptosystém *McEliece*, jak byl definován v [1]. Tento *kryptosystém* je první asymetrický šifrovací algoritmus založený na samoopravných kódech a je považován za hlavního představitele této kategorie šifer. Jako součást šifrování se používá úmyslného zanesení chyby do *zakódované* zprávy, čímž je informace přenášena ve zprávě ve své podstatě zničena, ale vlastník *soukromého klíče* je však schopný tyto chyby správným dekódováním odstranit a obnovit tak původní zašifrovanou zprávu.

Nejdříve v podkapitole 1.1 definujeme *klíče* a parametry *kryptosystému*, které potom použijeme v algoritmech pro *šifrování* a *dešifrování* v 1.2. Dále v podkapitole 1.3 popíšeme základní vlastnosti *kryptosystému* a nakonec v 1.4 zdůrazníme výpočetní problém, na kterém stojí základy *bezpečnosti* tohoto *kryptosystému*.

**Poznámka:** V této kapitole jsou použité základní termíny z oblasti kódování a obecné algebry, které jsou případně ujasněné v příloze B a respektive B. Též nadále předpokládáme operace s hodnotami z tělesa  $GF(2)$  – respektive s *bity*.

## 1.1 Generování klíčů

Generování potřebných klíčů je zajištěno následovně:

1. Zvolíme *lineární kód*  $\mathcal{K}$  s parametry  $(n, k, t)$  (opravující  $t$  chyb) a  $k \times n$  *generující maticí*  $G$ , pro který je znám efektivní dekódovací algoritmus<sup>1</sup>.
2. Vygenerujeme *náhodnou*  $k \times k$  *regulární matici*  $S$ .
3. Vygenerujeme *náhodnou*  $n \times n$  *permutační matici*  $P$ .
4. Vypočítáme  $k \times n$  matici  $\hat{G} = SGP$ .

Potom čísla  $k$ ,  $n$  a  $t$  jsou *veřejné parametry* systému, matice  $\hat{G}$  je *veřejný klíč* a kód s *generující maticí*  $G$  včetně matic  $S$  a  $P$  jsou *soukromým klíčem*.

**Poznámka:** Při generování klíčů je třeba vygenerovat regulární matici  $S$ . Pravděpodobnost, že náhodná čtvercová matice nad  $GF(2)$  je regulární, je přibližně 33 %. Toto tvrzení nebylo dokázáno, nicméně numerické výpočty tomu nasvědčují [20]. Pro získání této matice je tak v průměru potřeba vygenerovat 3 náhodné matice, což znamená  $3 \times n^2$  bitů. Efektivněji je možné matice generovat například dle [35].

## 1.2 Algoritmy pro šifrování a dešifrování

V této podkapitole uvedeme algoritmy pro šifrování a dešifrování tak, jak byly definovány *Robertem McEliece* v [1]. Na závěr podkapitoly uvedeme ověření platnosti dešifrování, neboli ukážeme, že dešifrovacím algoritmem je opravdu získána původní zašifrovaná zpráva.

### Šifrování

Šifrování zprávy  $m$  (o délce  $k$  bitů) veřejným klíčem  $\hat{G}$  probíhá následujícím způsobem:

1. Vygenerujeme náhodný vektor  $z$  délky  $n$  s *Hammingovou vahou*  $t^2$ .
2. Šifrovou zprávu  $c$  délky  $n$  sestojíme zakódováním generující maticí  $\hat{G}$  a přičtením chybového vektoru  $z$ .

$$c = m\hat{G} + z$$

---

<sup>1</sup> V původním článku [1] je kryptosystém definovaný pro libovolný *lineární kód* opravující zvolený počet chyb a jsou zmíněny *Goppa* kódy jako vhodný příklad k použití. Jak ukážeme dále, ne všechny lineární kódy jsou pro *McEliece* vhodné.

<sup>2</sup> V původním článku je uvedeno maximálně  $t$ , nicméně v pozdějších pracích na toto téma se uvádí právě  $t$ . Důvody jsou vysvětleny v kapitole 4.

**Dešifrování**

Obdrženou zašifrovanou zprávu  $c$  (délky  $n$ ) dešifrujeme následujícím způsobem:

1. Vypočítáme vektor  $\hat{c}$  délky  $n$ :  $\hat{c} = cP^{-1}$ .
2. Vektor  $\hat{c}$  dekódujeme zvoleným kódem na vektor  $\hat{m}$   
 $\hat{m} = \text{Dek}_{\mathcal{K}}(\hat{c})$
3. Vypočítáme původní zprávu  $m$ :  $m = \hat{m}S^{-1}$

**Ověření správnosti dešifrovacího algoritmu**

Správnost dešifrování můžeme ověřit následujícím způsobem:

- V prvním kroku dešifrovacího algoritmu je možné rozepsat původní zprávu  $m$ :

$$\hat{c} = cP^{-1} = (m\hat{G} + z)P^{-1} = (mSGP + z)P^{-1} = mSG + zP^{-1}$$

- Zavedeme substituci  $\hat{m} = mS$  a  $\hat{z} = zP^{-1}$ , potom

$$\hat{c} = mSG + zP^{-1} = \hat{m}G + \hat{z}$$

Z poslední rovnosti je vidět, že dekódováním je získán vektor  $\hat{m}$ , neboť  $\hat{z}$  je vektor s *Hammingovou vahou* maximálně  $t$  (matice  $P$  jen přehází jednotlivé bity vektoru  $z$ ).

$$\text{Dek}_{\mathcal{K}}(\hat{c}) = \hat{m}$$

- V posledním kroku stačí opět dosadit výše použitou substituci:

$$\hat{m}S^{-1} = mSS^{-1} = m$$

Dešifrováním je tedy získána původní zpráva  $m$ .

## 1.3 Základní vlastnosti kryptosystému

V této kapitole probereme základní fakta a vlastnosti *kryptosystému*. Popíšeme způsoby uložení a velikost klíčů a hlavní výhody a nevýhody použití *McEliece*.

### 1.3.1 Předpočítané matice

Je vidět, že původní matice  $S$  a  $P$  se ve výpočtu nepoužívají a pro dešifrování jsou potřeba pouze jejich *inverze*. Je tedy možné tyto matice předpočítat a *soukromý klíč* je tak trojice kód s generující maticí  $G$ , matice  $S^{-1}$  a matice  $P^{-1}$ .

### 1.3.2 Velikost klíčů

Největší nevýhodou *kryptosystému McEliece* je velikost klíčů. Již v původním článku jsou navrhovány parametry  $n = 1024$ ,  $k = 524$  a  $t = 50$ .<sup>3</sup> Za použití těchto parametrů má matice  $S$  (respektive její inverze) 274576 b  $\approx$  268 kb a (inverze) matice  $P$  1048576 b = 1 Mb.

Matice  $P$  je ve skutečnosti velmi *řídka* – každý *řádek* (respektive i *sloupec*) obsahuje pouze jednu jedničku, jinak je nulová. Je to permutační matice a lze ji reprezentovat prostou *permutací*, tedy ve formě  $\log_2 n$   $n$ -bitových indexů. Pro výše zmíněné hodnoty je to 10240 b = 10 kb.

Při konkrétním použití doporučených *binárních Goppa kódů* s těmito parametry je potřeba k uložení informace o použitém kódu  $\approx$  26 kb. Celkem se jedná o přibližně 300 kb dat pro uložení soukromého klíče

Pro uložení *veřejného klíče* (matice  $\hat{G}$ ) je třeba 536576 b = 524 kb dat.

Metody snížení velikosti klíčů *kryptosystému McEliece* jsou jedním z hlavních překážek pro rozšíření algoritmu a také jedním z hlavních cílů zkoumání tohoto *kryptosystému* a věnujeme se jim v kapitole 5.1.

### 1.3.3 Rychlost algoritmů

Naopak jednou z největších výhod algoritmu *McEliece* je rychlost algoritmů pro šifrování i dešifrování. Šifrování je prosté násobení matice s vektorem, což je jednoduchá operace, kterou je navíc možné provádět paralelně či efektivně implementovat v hardwaru. Dešifrování používá též násobení matic, ale složitější operace je dekodování vektoru  $\hat{m}$ . Viz kapitola 4.2 a tabulka 4.3.

## 1.4 Bezpečnost kryptosystému

Již v původním článku [1] *McEliece* zmiňuje dva možné útoky na navržený kryptosystém.

1. získání *soukromého* klíče ze znalosti *veřejného*
2. získání  $m$  bez nutnosti znát *soukromý* klíč

Nicméně je dobré již na tomto místě zmínit, že existují útoky využívající strukturu použitého kódu (tomuto tématu se věnuje kapitola 4.1.1).

### 1.4.1 Získání soukromého klíče

U prvního způsobu je v článku zmíněno, že je třeba rozložit  $\hat{G}$  na  $G$ ,  $S$  a  $P$ . Matici  $\hat{G}$  je sice možné dekomponovat v polynomiálním čase, ale množství

---

<sup>3</sup> Jak bude zmíněno dále, velikost těchto parametrů je pro dnešní použití nedostatečná.



jednotlivých matic je pro velká  $n$  a  $k$  obrovské, a získat tak původní matici hrubou silou je *neschůdné*<sup>4</sup>.

### 1.4.2 Získání původní zprávy

Druhý způsob znamená dekódovat původní zprávu  $m$  z přijaté zprávy  $c$ , která navíc obsahuje chybový vektor. Provést toto dekódování bez znalosti použitého kódu je *NP-těžký* problém [3].

#### Naznačení problému

V případě, že by byl chybový vektor *nulový*, platila by rovnost  $c = m\hat{G}$ . Výběrem  $k$  *dimenzí* (množina dimenzí  $\mathcal{D} \subset \{1, 2, \dots, n\}$  mající  $k$  prvků) vznikne  $\hat{G}_{\mathcal{D}}$  a  $c_{\mathcal{D}}$  z matice  $\hat{G}$  respektive vektoru  $c$ . Pokud je  $\hat{G}_{\mathcal{D}}$  regulární, lze řešit soustavu  $k$  nerovnic pro  $k$  neznámých  $(m_i)$  v polynomiálním (!) čase  $O(k^3)$ :

$$c_{\mathcal{D}} = m\hat{G}_{\mathcal{D}}$$

Za použití šifrovacího algoritmu *McEliece* je vektor  $c$  „zakrytý“ náhodným chybovým vektorem z *Hammingovy váhy*  $t$ . Potom pravděpodobnost, že  $c_{\mathcal{D}}$  (ve výběru  $k$  dimenzí) je bez chyby je  $(1 - \frac{t}{n})^k$  [1]. Pro  $O(k^3)$  operací pro vyřešení jedné soustavy rovnic je to přibližně:

$$O\left(\frac{n^3}{(1 - \frac{t}{n})^k}\right) = O\left(n^3 \left(\frac{n}{n-t}\right)^k\right)$$

Zlomek  $\frac{n}{n-t}$  je jistě větší než 1, tudíž pro velká  $k$  výrazně převyšuje druhý činitel a jedná se o *NP-těžký* problém.

Navíc není jasné,  *které* z nalezených řešení odpovídá původní zprávě  $m$ .

<sup>4</sup> Např. jen počet možných *permutačních matic* je  $n!$ . Počet *generujících matic* závisí na zvoleném kódu.



## Elektronický podpis

V původním článku od *Roberta McEliece* [1] bylo zmíněno, že takto navrženým kryptosystémem nelze získat schéma pro *elektronický podpis*. Původní algoritmy byly navrženy pouze pro *asymetrické šifrování*. Až v roce 2001 byl v [13] publikován postup pro získání elektronického podpisu za pomoci asymetrického kryptosystému založeného na samoopravných kódech.

Jak ukážeme níže, pro získání elektronického podpisu se bude více hodit kryptosystém *Niederreiter*, který je ve své podstatě variantou kryptosystému *McEliece*.

### 2.1 Kryptosystém Niederreiter

V roce 1986 publikoval *Harald Niederreiter* v [31] kryptosystém s veřejným klíčem využívající stejných principů jako kryptosystém *McEliece*. Tento kryptosystém je též založený na *lineárních kódech* a jeho bezpečnost též stojí na problému dekódování neznámého kódu. Na rozdíl však od kryptosystému *McEliece* používá k sestrojení klíčů *kontrolní* matici místo matice *generující*.

#### 2.1.1 Generování klíčů

Generování klíčů probíhá následovně:

1. Zvolíme *lineární kód*  $(n, k)$ , opravující  $t$  chyb s odpovídající  $(n - k) \times n$  *kontrolní maticí*  $H$ .
2. Vygenerujeme *náhodnou*  $(n - k) \times (n - k)$  *regulární matici*  $S$ .
3. Vygenerujeme *náhodnou*  $n \times n$  *permutační matici*  $P$ .
4. Vypočítáme  $(n - k) \times n$  matici  $\hat{H} = SHP$ .

Potom čísla  $k$ ,  $n$  a  $t$  jsou *veřejné parametry* systému, matice  $\hat{H}$  je *veřejný klíč* a kód s *kontrolní maticí*  $H$  a matice  $S$  a  $P$  jsou *soukromým klíčem*.

### 2.1.2 Algoritmy pro šifrování a dešifrování

V této podkapitole uvedeme algoritmy pro šifrování a dešifrování z [31] a ověření správnosti dešifrovacího algoritmu.

#### Šifrování

Šifrování zprávy probíhá následujícím způsobem:

1. Máme zprávu  $m$  dlouhou  $n$  bitů s *Hammingovou vahou* maximálně  $t$ . Tato zpráva reprezentuje *chybový vektor* pro použitý kód.
2. Šifrový text  $c$  (délky  $n-k$ ) spočteme jako *syndrom* zprávy  $m$  (respektive chyby) za použití matice  $\hat{H}$ :  $c = m\hat{H}^T$ .

**Poznámka:** Chybový vektor  $m$  požadované délky  $n$  a *Hammingovy váhy*  $t$  lze získat *zakódováním*<sup>5</sup> původní zprávy  $k$  zašifrování. Je vidět, že možných zpráv je pro  $t \ll n$  řádově méně než všech možných vektorů délky  $n$ . Způsob zakódování bude probírán níže při popisu získání *elektronického podpisu* pomocí tohoto *kryptosystému*.

#### Dešifrování

Obdržená šifrová zpráva  $c$  se dešifruje následujícím způsobem:

1. Vypočteme vektor  $\hat{c}$  délky  $n-k$ :  $\hat{c} = c \left(S^T\right)^{-1}$
2. Pomocí dekodovacího algoritmu použitého kódu získáme z  $\hat{c}$  chybový vektor  $\hat{m}$  (délky  $n$ ).
3. Původní zprávu  $m$  získáme výpočtem  $m = \hat{m} \left(P^T\right)^{-1}$

**Poznámka:** Stejně jako je tomu u *kryptosystému McEliece*, je možné hodnoty  $\left(P^T\right)^{-1}$  a  $\left(S^T\right)^{-1}$  předpočítat. Navíc inverzi  $P$  je opět možné uložit jako  $\log_2 m$   $n$ -bitových hodnot, jelikož se jedná o permutaci. Soukromý klíč je tak trojice kód s kontrolní maticí  $H$ , matice  $\left(P^T\right)^{-1}$  a matice  $\left(S^T\right)^{-1}$ .

#### Ověření správnosti dešifrovacího algoritmu

Postup ověření správnosti dešifrování je následující:

- V prvním kroku dešifrovacího algoritmu je možné výpočet rozepsat následujícím způsobem:

$$\hat{c} = c \left(S^T\right)^{-1} = m\hat{H}^T \left(S^T\right)^{-1} = mP^T H^T S^T \left(S^T\right)^{-1} = mP^T H^T$$

---

<sup>5</sup> Zde nejsou na mysli samoopravné kódy, ale pouze jednoznačné zakódování zprávy.

- Zavedeme substituci  $\hat{m} = mP^T$ , potom  $\hat{c} = \hat{m}H^T$ , což odpovídá výpočtu *syndromu* pro použitý kód. Jelikož  $\hat{m}$  je pouze *permutovaná* původní  $m$ , má *Hammingovu váhu*  $t$  a pomocí dekodovacího algoritmu získáme  $\hat{m}$  jako *chybový vektor*.
- Nakonec se jen  $\hat{m}$  vynásobí inverzí matice  $P^T$

$$\hat{m} \left(P^T\right)^{-1} = mP^T \left(P^T\right)^{-1} = m$$

Dešifrováním jsme tedy získali původní zprávu  $m$ .

### 2.1.3 Vlastnosti kryptosystému

Kryptosystém *Niederreiter* je variantou asymetrického kryptosystému založeného na lineárních kódech, podobně jako kryptosystém *McEliece*. Šifrovým textem není zakódované slovo, jak je tomu u *McEliece*, nýbrž *syndrom* chybového vektoru, který je možné dekodovat pouze za znalosti skrytého lineárního kódu.

V [43] byla dokázána ekvivalence složitosti prolomení tohoto kryptosystému s kryptosystémem *McEliece*. Útočník, který dokáže prolomit jeden ze systémů dokáže prolomit i druhý. Další informace jsou k nalezení v [31, 13].

## 2.2 Schéma pro elektronický podpis

V roce 2001 autoři *Courtois a spol.* v [13] publikovali postup, jakým lze získat z kryptosystému založeném na lineárních kódech schéma pro *elektronický podpis*. Autoři zmiňují, že je možné stejným způsobem využít i kryptosystém *McEliece*, nicméně kvůli délce výsledného *podpisu* je mnohem praktičtější využít kryptosystém *Niederreiter*.

Pro úplnost zde uvedeme původní argumenty, které zmínil *Robert McEliece* v [1] a které bránily v použití kryptosystému *McEliece* jako algoritmu pro digitální podpis.

### 2.2.1 Překážky pro použití McEliece pro podepisování

Abychom mohli využít algoritmus pro dešifrování jako algoritmus *podepisování*, bylo by potřeba, aby vektor  $c$  (resp.  $\hat{c}$ ) bylo možné dekodovat na kódové slovo. Nicméně pro původně navrhované parametry je poměr počtu vektorů délky  $n$  v *Hammingově vzdálenosti*  $t$  od kódových slov ku všem vektorům délky  $n$  téměř nulový. Takový algoritmus pro podepisování by prakticky vždy selhal a nebylo by možné získat žádný výstup jako *podpis*.

Konkrétně pro navrhované parametry  $n = 1024$ ,  $t = 50$  (a  $k = 524$ ) je počet vektorů do *Hammingovy vzdálenosti* 50 od všech kódových slov:

$$2^{524} \sum_{i=0}^{50} \binom{1024}{i} \approx 2^{808}$$

Počet všech vektorů délky 1024 je  $2^{1024}$ . Tedy pravděpodobnost, že vektor délky 1024 půjde algoritmem *dekódovat* je přibližně  $2^{-216}$  [1].

Algoritmus *Niederreiter* selhává naprosto stejným způsobem [13].

### 2.2.2 Vyhovující parametry

Autoři *Courtois a spol.* v článku [13] dokázali vzorec pravděpodobnosti, že náhodný *syndrom* délky  $n - k$  (a při použití *Goppa kódů*) je možné dekodovat je

$$\mathcal{P} = \frac{N_{\text{dekódovatelné}}}{N_{\text{celkem}}} \approx \frac{\frac{n^t}{t!}}{n^t} = \frac{1}{t!}$$

A závisí tedy pouze na počtu chyb  $t$ . V článku je popsána volba parametrů<sup>6</sup> a pro bezpečnost odpovídající 80 bitům symetrické šifry jsou zvoleny parametry  $n = 2^{16}$  a  $t = 9$ . Pravděpodobnost, že pro zadané parametry bude náhodný vektor možné dekodovat jako *syndrom* je  $\frac{1}{9!} \approx 2^{-19}$ . Pro získání platného *syndromu* bude tedy nutné v průměru vygenerovat  $2^{19}$  vektorů.

### 2.2.3 Popis schématu

Dle kapitoly výše je nutné získat několik ( $9!$ ) vektorů k odpovídajícímu *dokumentu*, který je třeba *podepsat*. To je možné zajistit jednoduše použitím *hashovací* funkce  $h$  s tím, že je společně s dokumentem hashován i náhodný index  $i$ . Ten je možné postupně zvyšovat, dokud výstup  $h$  nebude možné *dekódovat* a získat odpovídající chybový vektor  $z$ . Jak ukážeme dále, hodnota  $i$  bude třeba pro ověření podpisu a je nutné tuto hodnotu k podpisu připojit.

#### Značení

Nechť  $h$  je kryptograficky bezpečná *hashovací* funkce, jejíž výstup je dlouhý přesně  $n - k$  bitů. Dále  $D$  je dokument, který je třeba *podepsat* a  $s = h(D)$  *hash* (*otisk*) dokumentu. Zřetězení  $s$  a  $i$  bude značeno jako  $(s|i)$  a  $s_i = h(s|i)$  je tedy *otisk* dokumentu za použití odpovídajícího *indexu*  $i$ . Nejmenší  $i$  takové, že  $s_i$  lze dekodovat, bude značeno  $i_0$ . Odpovídající  $s_{i_0}$  je tedy *syndrom*, který bude použitý pro podpis  $D$ . Nakonec chybový vektor  $z$  odpovídá *syndromu*  $s_{i_0}$  a podpis  $S$  je tedy dvojice  $S = (z|i_0)$

#### Délka podpisu

Délka podpisu závisí na uložení dat  $z$  a  $i_0$ . Vektor  $z$  je chybový vektor odpovídajícího samoopravného kódu. Jeho *Hammingova váha* je  $t$  a je tedy velmi řídký. Existuje pouze  $\binom{n}{t}$  vektorů *váhy*  $t$  a délky  $n$  a je tedy možné tento řídký vektor komprimovat. V [13] je uvedeno, jak všechny možné vektory seřadit a vyjádřit tak konkrétní vektor pouze jeho *indexem*  $I_z$ . Takový *index* je pak možno uložit v  $\log_2 \binom{n}{t}$  bitech.

---

<sup>6</sup> S ohledem na útok *Canteaut-Chabaud* [11].

Index  $i_0$  bude zabírat v průměru  $\log_2 t!$  bitů a nelze ho uložit žádným kompaktnějším způsobem.

Pro konkrétní uvedený příklad ( $n = 2^{16}$ ,  $t = 9$ ) je pak průměrná velikost podpisu  $S = (I_Z|i_0) : \log_2 \binom{2^{16}}{9} + \log_2 9! = 125.5 + 18.4 = 144$  b.

## 2.3 Algoritmy schématu pro digitální podpis

### Algoritmus pro podepisování

Podpis sestojíme následujícím způsobem:

- Vypočítáme *hash*  $s$  dokumentu  $D$ :  $s = h(D)$ .
- Nalezneme nejmenší  $i$  ( $i_0$ ) takové, že  $s_i = h(s|i)$  lze dekodovat.
- Použijeme algoritmus *Niederreiter* pro dešifrování k nalezení chybového vektoru  $z$ , že  $z\hat{H}^T = s_{i_0}$
- Převědeme  $z$  na index  $I_z$ .
- Použijeme  $S = (I_Z|i_0)$  jako podpis dokumentu  $D$ .

### Algoritmus pro ověření

Ověření přijaté zprávy  $D$  s podpisem  $S = (I_Z|i_0)$  provedeme tímto postupem:

- Převědeme index  $I_z$  zpět na vektor  $z$ .
- Spočítáme  $s_1 = z\hat{H}^T$  pomocí veřejného klíče  $\hat{H}$
- Spočítáme *hash*  $s_2 = h(h(D)|i_0)$
- Pokud se  $s_1$  a  $s_2$  shodují, podpis je platný.

Důkaz tohoto ověření je prostý. „Zašifrováním“  $z$  získáme původní  $s_{i_0}$ , které – v případě platného podpisu – odpovídá zahashované zprávě  $D$  společně s indexem  $i_0$ .

**Poznámka:** Bezpečnost schématu pro elektronický podpis závisí na jedno-směrné funkci dekódování syndromu. Tuto operaci není možné provést bez znalosti *soukromého klíče* – matic  $H$ ,  $S$  a  $P$  [31, 43].

V případě použití kryptosystému *McEliece* pro získání podpisu, bychom ve třetím kroku algoritmu pro podepisování místo *syndromu* slovo délky  $k$ . Při zvolených parametrech ( $n = 2^{16}$  a  $t = 9$ ) je  $k$  rovno  $2^m - mt = 2^{16} - 16 \cdot 9 = 64$  kb, což je velikost pro podpis prakticky nepřijatelná (často by byl podpis delší než původní *dokument*).





## Binární Goppa kódy

Pro použití kryptosystému *McEliece* je potřeba *lineárního* kódu, který dokáže opravit zvolený počet chyb a pro který je znám *efektivní* (polynomiální) dekodovací algoritmus. V této kapitole uvedeme *binární Goppa kódy*, které tuto podmínku splňují a navíc dlouhodobě odolávají útokům na strukturu kódu, kterým se věnujeme v kapitole 4. Většina moderních variant *kryptosystému* vychází z těchto kódů a proto je důležité se jimi zabývat.

Novou kategorii *lineárních* kódů definoval v roce 1970 *Valery Goppa* v [19]. Tyto kódy byly později pojmenovány po svém autorovi a první anglicky psaný článek na téma *Goppa* kódů publikoval *Elwyn Berlekamp* v roce 1973. V této podkapitole uvedeme definice a algoritmy nutné pro použití *Goppa* kódů, které jsou k nalezení v [4, 15]. Další informace o těchto kódech jsou k nalezení například v [27].

### Poznámky

V této kapitole vycházíme ze základních poznatků z oblasti samoopravných kódů, které jsou uvedené v příloze B. Použité značení v kapitole odpovídá značení, které se používá ve zmíněné příloze.

*Binární Goppa kódy* využívají struktury z obecné algebry, především tzv. *konečná* a *rozšířená tělesa*. Teorie z této oblasti je uvedena v příloze A. Dále budeme používat termín *rozšířené těleso* ve smyslu konečného tělesa reprezentované polynomy s koeficienty z tělesa  $GF(2^m)$ . Rozšířením tak získáme těleso  $GF(2^m)^n$ .<sup>7</sup>

Obecné *Goppa* kódy jsou definovány pomocí *algebraických křivek*<sup>8</sup>, nicméně v této práci se budeme zabývat pouze podkategorií, tzv. *binárními Goppa kódy*.

<sup>7</sup> Toto těleso je izomorfní s tělesem  $GF(2^{mn})$ .

<sup>8</sup> *Goppa* kódy jsou též nazvány jako *algebraické geometrické (AG) kódy*.

### 3.1 Sestrojení Goppa kódu

Nechť existuje polynom  $g$  z okruhu polynomů nad konečným tělesem  $GF(2^m)$  stupně  $t$  a posloupnost  $L$   $n$  navzájem různých prvků z  $GF(2^m)$ , které zároveň nejsou kořeny polynomu  $g$ .

$$g \in \mathbb{F} = GF(2^m)[x]$$

$$L = (L_1, \dots, L_n), \forall i, j : L_i \in \mathbb{F} \wedge L_i \neq L_j \wedge g(L_i) \neq 0$$

Pak *binární Goppa* kód (prostor kódových slov)  $\Gamma$  definujeme:

$$\Gamma(g, L) = \left\{ c \in GF(2^n) \mid \sum_{i=1}^n \frac{c_i}{x - L_i} \equiv 0 \pmod{g(x)} \right\}$$

Polynom  $g(x)$  nazýváme *Goppův* polynom a  $n$ -tici  $L$  *podporou* kódu<sup>9</sup>.

Takto sestrojený kód má parametry  $(n, k, t) = (n, 2^m - tm, t)$

**Poznámka:** U *binárních Goppa* kódů je polynom  $x - L_i$  prvek  $(0 \dots 01)(L_i)$ . Důvod podmínky  $g(L_i) \neq 0$  je tak jasně vidět z definice, protože musí existovat inverze tohoto prvku. Důvod druhé podmínky – vzájemně různé prvky  $L_i$  – bude vidět později, ale podobně jako u *Hammingových* kódů dle sloupcového vektoru matice  $H$  zjišťujeme pozici, kde nastala chyba, tak u *Goppa* kódů budeme zjišťovat pozici dle prvků  $L_i$  a proto se také jedná o posloupnost, nikoliv o množinu.

#### 3.1.1 Ireducibilní binární Goppa kódy

Pokud je  $g$  *ireducibilní*, nazveme  $\Gamma$  *ireducibilním binárním Goppa* kódem. V tomto případě může mít množina  $L$  až  $n = 2^m$  prvků, neboť *ireducibilní* polynom nemá žádné kořeny a tak pro všechny  $a \in GF(2^m)$  (včetně  $0$ ) platí podmínka  $g(L_i) \neq 0$ . Takový kód má tedy parametry  $(n, k, t) = (2^m, 2^m - mt, t)$ , a jsou tedy jednoznačně určené parametrem  $m$  („velikostí vnitřního tělesa“) a stupněm polynomu  $g$ , neboli počtem opravitelných chyb  $t$ .

---

<sup>9</sup> Anglicky *Goppa polynomial*  $g$  a *support*  $L$ .

### 3.1.2 Sestrojení kontrolní matice

Z definice lze sestrotit *kontrolní* matici  $H$  v následujícím tvaru (detailní postup sestrotění matice lze nalézt např. v [23]):

$$H = \begin{pmatrix} (g_t)g(L_1)^{-1} & \dots & (g_t)g(L_n)^{-1} \\ (g_{t-1}L_1g_t)g(L_1)^{-1} & \dots & (g_{t-1}L_ng_t)g(L_n)^{-1} \\ \vdots & \ddots & \vdots \\ (g_1 + L_1g_2 + \dots + L_1^{t-1}g_t)g(L_1)^{-1} & \dots & (g_1 + \dots + L_n^{t-1}g_t)g(L_n)^{-1} \end{pmatrix}$$

a tuto matici lze vyjádřit jako součin matic  $H = KVD$ , kde  $K$  je matice koeficientů polynomu  $g$ ,  $V$  je tzv. *Vandermondova* matice a  $D$  je diagonální matice:

$$K = \begin{pmatrix} g_t & 0 & \dots & 0 \\ g_{t-1} & g_t & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ g_1 & g_2 & \dots & g_t \end{pmatrix} \quad V = \begin{pmatrix} 1 & 1 & \dots & 1 \\ L_1 & L_2 & \dots & L_n \\ \vdots & \vdots & \ddots & \vdots \\ L_1^{t-1} & L_2^{t-1} & \dots & L_n^{t-1} \end{pmatrix}$$

$$D = \begin{pmatrix} g(L_1)^{-1} & & & \\ & g(L_2)^{-1} & & \\ & & \ddots & \\ & & & g(L_n)^{-1} \end{pmatrix}$$

Matice  $K$  je regulární ( $g_t \neq \mathbf{0}$  a řádky jsou tedy jistě *lineárně nezávislé*), existuje tedy  $K^{-1}$ . Z definice kontrolní matice  $GH^T = \mathbf{0}$  můžeme tedy sestrotit jednodušší kontrolní matici:

$$\begin{aligned} GH^T &= G(KVD)^T = \mathbf{0} \\ G(KVD)^T (K^T)^{-1} &= \mathbf{0} (K^T)^{-1} \\ G(VD)^T K^T (K^T)^{-1} &= \mathbf{0} \\ G(VD)^T &= \mathbf{0} \end{aligned}$$

Matice  $VD$  tedy splňuje definici *kontrolní* matice a navíc je jednodušší na sestrotění než  $KVD$ . Proto *kontrolní* matici *binárního Goppa kódu* definujeme jako  $H = VD$ .

$H$  je  $t \times n$  matice nad tělesem  $GF(2^m)$ .  $H$  nad  $GF(2)$  získáme jednoduše „rozbalením“ prvků  $GF(2^m)$  do sloupcových vektorů  $m$  bitů. Z každé řádky nad  $GF(2^m)$  tedy vznikne  $m$  řádků nad  $GF(2)$  a tato *kontrolní* matice  $H$  pak má rozměry  $r \times n = (mt) \times (2^m)$ . *Generující*  $k \times n$  matici  $G$  získáme klasickým převodem těchto matic u *lineárních kódů*<sup>10</sup>.

<sup>10</sup> Pro úplnost tento postup uvádíme v příloze B.2.

## 3.2 Dekódování

Pro dekódování, respektive opravu chyb, existuje několik algoritmů. V této kapitole uvedeme *Pattersonův algoritmus*, který byl představen *Nicholasem Pattersonem* v roce 1975 v [34]. Další algoritmy pro dekódování algoritmů – především tzv. *List Decoding* algoritmy – se dají nalézt v [36, 7].

### 3.2.1 Pattersonův algoritmus

*Syndrom* přijatého slova  $c$  je možné počítat z kontrolní matice  $H$  nebo též jako polynom z definice kódu  $\Gamma$ :

$$s(x) \equiv \sum_{i=1}^n \frac{c_i}{x - L_i} \pmod{g(x)}$$

Takto spočítaný syndrom jistě závisí pouze na chybovém vektoru  $e$ :

$$\begin{aligned} \sum_{i=1}^n \frac{c_i}{x - L_i} &= \sum_{i=1}^n \frac{b_i}{x - L_i} + \sum_{i=1}^n \frac{e_i}{x - L_i} \\ \sum_{i=1}^n \frac{b_i}{x - L_i} + \sum_{i=1}^n \frac{e_i}{x - L_i} &\equiv \sum_{i=1}^n \frac{e_i}{x - L_i} \pmod{g(x)} \end{aligned}$$

Pokud je  $s$  nulový, přijali jsme kódové slovo a zprávu  $d$  můžeme získat výběrem patřičných dimenzí (dle matice  $G$ ). Pokud  $s$  není nulový vektor, provedeme následující kroky pro opravení vzniklých chyb:

1. Vypočítáme  $r(x) = \sqrt{x - s(x)^{-1}}$  v tělese určeném polynomem  $g$ .
2. Rozložíme  $r$  na polynomy  $\alpha$  stupně  $\leq \lfloor \frac{t}{2} \rfloor$  a  $\beta$  stupně  $\leq \lfloor \frac{t-1}{2} \rfloor$  tak, že:

$$\alpha(x) \equiv \beta(x)r(x) \pmod{g(x)}$$

3. Sestrojíme polynom  $\sigma = \alpha^2 + x\beta^2$ , tzv. *lokátor chyb*.
4. Kořeny  $L_i$  (z podpory  $L$ ) polynomu  $\sigma$  odpovídají chybám na pozici  $i$ .
5. Z nalezených kořenů sestrojíme chybový vektor  $e$  a opravíme přijaté slovo  $c$  standardním způsobem  $c' = c + e$ .

Odvození tohoto algoritmu je možné nalézt v [34]. Poznámky k jednotlivým krokům algoritmu uvádíme níže.

#### Předpočítané dílčí syndromy

Pro výpočet syndromu přijatého slova je nutné počítat výraz  $(x - L_i)^{-1}$  (pro každý prvek podpory  $L_i$ , kde odpovídající bit  $c_i$  je rovný 1). Tento výraz je stejný pro dekódování *každého* slova a je tedy vhodné si tyto tzv. *dílčí syndromy* předpočítat a uložit je v rámci parametrů určující daný kód  $\Gamma$ .

**Výpočet odmocniny**

Odmocninu v rozšířeném binárním tělese můžeme jednoduše odvodit. Prvek  $r$  je *odmocninou* prvku  $b$  (z tělesa  $\mathbb{F}$ , pokud platí:

$$r^2 = a \quad \Rightarrow \quad r = \sqrt{a}$$

Nechť  $N$  je počet prvků *multiplikativní grupy* tělesa  $\mathbb{F}$ , potom položíme  $r = a^{\frac{N+1}{2}}$  a umocníme  $r$  na druhou:

$$r^2 = a^{\frac{N+1}{2} \cdot 2} = a^{N+1} = a^N \cdot a^1$$

Dle *Lagrangeovy* věty platí, že  $a^N = 1$  a tudíž zvolené  $r$  (pokud existuje) je právě hledaná odmocnina. V tělese s charakteristikou 2 je počet prvků vždy lichý (například v rozšířeném tělese je to  $2^{mt} - 1$ ) a tudíž zlomek  $\frac{N+1}{2}$  dává smysl a odmocninu lze vypočítat jako mocninu:

$$\sqrt{a} = a^{\frac{(2^{mt}-1)+1}{2}} = a^{2^{mt-1}}$$

**Rozložení polynomu**

Rozložení polynomu  $r$  na  $\alpha$  a  $\beta$  je ve skutečnosti snadné. Rovnice  $\alpha(x) \equiv \beta(x)r(x) \pmod{g(x)}$  je rovnicí, která vzniká při výpočtu *rozšířeného Euklidova algoritmu*. Polynom  $\alpha$  odpovídá „zbytku“ a  $\beta$  „koeficientu“ při výpočtu *EEA* (viz příklad *EEA* v kapitole 6.1.3.4). Polynomy požadovaného stupně získáme zastavením výpočtu *EEA* přesně v polovině, respektive v kroku, kdy stupeň „zbytku“ klesne pod  $\lfloor \frac{t}{2} \rfloor$ .

**Nalezení kořenů polynomu**

Tento krok algoritmu je dle [15] asymptoticky nejnáročnější. Základní způsob pro nalezení kořenů polynomu je hrubou silou vypočítat hodnotu  $\sigma(L_i)$  pro všechny koeficienty  $L_i$  z podpory  $L$ . Toto dosazení do polynomu lze provést pomocí tzv. *Hornerova* schématu.

Efektivnější algoritmem je tzv. *Chienův způsob* hledání kořenů<sup>11</sup> [44, 20], který využívá výpočtu kořenů pomocí primitivního prvku  $\alpha$ . Pro všechny prvky  $\alpha^i$  tělesa  $GF(2^m)$  (kromě nulového prvku) platí:

$$\begin{aligned} \sigma(\alpha^i) &= \sigma_s \cdot (\alpha^i)^s & + \dots + \sigma_1 \cdot (\alpha^i) & + \sigma_0 = \\ &= \gamma_{s,i} & + \dots + \gamma_{1,i} & + \gamma_0 \\ \sigma(\alpha^{i+1}) &= \sigma_s \cdot (\alpha^{i+1})^s & + \dots + \sigma_1 \cdot (\alpha^{i+1}) & + \sigma_0 = \\ &= \alpha \cdot \gamma_{s,i} & + \dots + \alpha \cdot \gamma_{1,i} & + \gamma_0 \end{aligned}$$

<sup>11</sup> Tento algoritmus byl původně navržený pro *BCH* kódy v [12]

### 3. BINÁRNÍ GOPPA KÓDY

---

Z posledního řádku je vidět, jak lze tohoto faktu při výpočtu všech kořenů využít. Vypočteme-li tedy hodnotu  $\sigma(\alpha)$ , tak další hodnoty  $\sigma(\alpha^i)$  vypočteme pomocí  $s - 1$  operací násobení a  $s - 1$  sčítání  $\Rightarrow O(s)$  násobení.

Druhým způsobem, jak nalézt kořeny polynomu  $\sigma$  je *rozložení* tohoto polynomu na faktory ve tvaru  $(x - L_i)$ . Toho se dá docílit *Berkleampovým* algoritmem [5]. Nalezení pozic chyb je pak pouhým vyhledáním získaných  $L_i$  v posloupnosti  $L$ .

# Kryptoanalýza systému McEliece

Již v původním článku [1] byly naznačeny 2 aspekty, díky kterým je možné považovat kryptosystém *McEliece* za *bezpečný*:

1. Problém nalezení kódového slova *obecného lineárního kódu* s minimální vzdáleností k danému vektoru – *problém obecného dekódování* – je *NP-těžký* [3]
2. Není znám žádný algoritmus, který by *bez znalosti tajných parametrů* dokázal nalézt kódové slovo efektivněji, než *za použití obecného kódu*.

Druhý z těchto aspektů neplatí za použití libovolného kódu, jak bude ukázáno v kapitole 4.1.1. Při použití některých lineárních kódů je možné odhalit strukturu použitého kódu.

I přes tato tvrzení je nutné zvolit parametry  $n$ ,  $k$  a  $t$  tak, aby útok hrubou silou byl časově (a případně i prostorově) neschůdný. Volbu bezpečných parametrů probíráme v kapitole 4.2.

## 4.1 Útoky na McEliece

V této podkapitole uvedeme některé z útoků na kryptosystém *McEliece*. Dle [15] se útoky dají rozdělit do dvou hlavních kategorií:

- útoky na soukromý klíč
- útoky na šifrový text

Do první kategorie spadají útoky na strukturu použitého kódu a *Support Splitting Algorithm* [37]. Jedná se o útoky, ve kterých útočník ze znalosti *veřejného klíče* sestrojí klíč *soukromý*. Do druhé kategorie spadají útoky, které

nezjišťují *soukromý klíč*, ale z *šifrovaného textu* odhalují text *otevřený*. To zahrnuje *útok s informační množinou*, navržený již Robertem McEliece, *nalezení kódového slova s nízkou vahou* a další útoky na kryptosystém *McEliece*.

Nerozumné použití kryptosystému vede k možnému zneužití několika *slabin*, které jsou probrány ve zvláštní podkapitole 4.3<sup>12</sup>.

#### 4.1.1 Útoky na strukturu použitého kódu

V historii byly zaznamenány pokusy o sestrojení *soukromého klíče* za použití jiných lineárních kódů než *Goppa kódů*. Tyto návrhy vznikají hlavně kvůli zredukování velikosti klíčů, které jsou za použití *Goppa kódů* obrovské. Většina z těchto návrhů ale byla shledána jako nedostatečně bezpečná pro použití v asymetrické kryptografii.

V původním článku, kde byl definován kryptosystém *Niederreiter*, bylo navrženo použití *zobecněných Reed-Solomon (GRS) kódů* [31]. V [39] bylo prokázáno, že je možné skrytou strukturu *GRS* kódu odhalit v polynomiálním čase. Stejně podmínky platí i pro použití v kryptosystému *McEliece*.

Použití tzv. *Alternantních* či dalších kódů, používajících kompaktní uložení klíčů bylo prolomeno *algebraickou* a *strukturální kryptoanalýzou* [16, 17, 42].

#### 4.1.2 Support Splitting Algorithm

Tento algoritmus, navržený Nicolasem Sendrier, dokáže v *polynomiálním čase* (přibližně  $O(n^4)$ ) určit, zda 2 lineární kódy jsou *permutačně ekvivalentní* [37].

**Definice 1** *Nechť existují dva lineární kódy  $K_1$  a  $K_2$ . Říkáme, že tyto kódy jsou permutačně ekvivalentní, pokud všechna kódová slova kódu  $K_1$  lze převést na kódová slova  $K_2$  použitím stejné permutace bitů (pozic)  $P$ .*

Pokud má útočník k dispozici *Goppa kód* (určený polynomem  $g$ ), dokáže v polynomiálním čase rozhodnout, jestli je permutačně ekvivalentní s kódem, který generuje *veřejný klíč*  $\tilde{G}$ . Pokud by bylo množství možných *Goppa polynomů* – resp. *Goppa kódů* – nízké, útočník by mohl hrubou silou odhalit použitý *Goppa kód*. Z tohoto důvodu je nutné, aby generované *Goppa polynomy* měly koeficienty z větších binárních těles. Čím větší budou vnitřní tělesa, tím více existuje možných (ireducibilních) polynomů a není tak možné projít všechny možnosti hrubou silou [36].

#### 4.1.3 Útok s informační množinou

*Útok s informační množinou* (*Information Set Decoding attack – ISD*) byl popsán již v původním článku *Roberta McEliece* [1] a zmíněn v kapitole 1.4. Později byl tento útok formalizován a zobecněn v [25].

---

<sup>12</sup> Nejedná se totiž o útoky na kryptosystém ale spíše o nepříjemné *vlastnosti* kryptosystému, se kterými je nutné počítat.



Útok je založen na výběru  $k$  sloupců – dimenzí – (množina  $K \subset \mathbb{N}_n$  s  $k$  prvky) z veřejně známé matice  $\hat{G}$  tak, aby vzniklá matice  $\hat{G}_K$  byla *regulární* a bylo možné vyřešit vzniklou soustavu rovnic

$$c_K = m\hat{G}_K$$

Tomuto útoku brání fakt, že útočník neví, které bity šifrovaného textu jsou (v průběhu šifrování) „zamaskované“ vygenerovaným náhodným vektorem  $z$ . Případný útočník tak zároveň musí vybrat dimenze takové, které nejsou zatížené tímto chybovým vektorem.

Autoři *Lee* a *Brickell* zobecnili tento útok tak, že není nutné vybrat množinu dimenzí, která neobsahuje chybu. Pokud bude množství chyb malé, je možné tento fakt do algoritmu započítat a bity vektoru  $c$  respektive  $c_K$  invertovat.

Pravděpodobnost, že výběr  $k$  dimenzí bude obsahovat maximálně  $j$  chyb je

$$\mathcal{P}_j = \frac{N_{\text{max. } j \text{ chyb}}}{N_{\text{celkem}}} = \frac{\sum_{i=0}^j \binom{t}{i} \binom{n-t}{k-i}}{\binom{n}{k}}$$

A počet všech vektorů  $z_K$ , jejichž *Hammingova váha* je menší než  $j$  (tedy počet vektorů, které je třeba vyzkoušet a zprávu  $c$  dle tohoto vektoru invertovat) je

$$N_j = \sum_{i=0}^j \binom{k}{i}$$

Pokud je možné řešit soustavu  $k$  lineárních rovnic v  $O(k^3)$  počtu krocích, je asymptotická složitost tohoto útoku

$$W_j = O\left(\mathcal{P}_j^{-1} \left(k^3 + kN_j\right)\right)$$

V průměru je totiž provést  $\mathcal{P}_j^{-1}$  výběrů dimenzí, pro každý výběr provést v průměru  $kN_k$  invertování bitů a nakonec vyřešit soustavu rovnic – pokud je řešitelná.

Autoři uvádí, že pro minimalizaci  $W_j$  je rozumné – při rozumných velikostech kódů – volit  $j = 2$ . Tento útok v době publikování snížil složitost útoku na *kryptosystém McEliece* přibližně  $2^1$ -krát [25].

#### 4.1.4 Nalezení kódového slova s nízkou vahou

Jako nejúspěšnější útok na nalezení tajné zprávy se v posledních letech jeví tzv. *útok nalezením slova s nízkou vahou*. Z definice šifrování je známo, že  $c$  leží ve vzdálenosti  $t$  od nějakého kódového slova. Sestrojíme nový kód  $K'$  s generující maticí  $\hat{G}'$  tak, že k matici  $\hat{G}$  přidáme šifrový text  $c$  jako další řádek matice

$$\hat{G}' = \begin{pmatrix} \hat{G} \\ c \end{pmatrix} = \begin{pmatrix} \hat{G} \\ m\hat{G} + z \end{pmatrix}$$

Původní kód generovaný maticí  $\hat{G}$  měl *kódovou vzdálenost* minimálně  $2t+1$  a nově vzniklý kód  $\mathcal{K}'$  má *kódovou vzdálenost*  $t$ . Navíc jediný vektor, s vahou  $t$  je neznámý chybový vektor  $z$  (který je potřeba k úspěšnému dekódování či útoku *ISD*).

Cílem tohoto útoku je tedy nalézt kódové slovo  $z$  (s nejnižší vahou) z výše definovaného kódu  $\mathcal{K}'$ . Algoritmy představené v [26, 40, 11] nejdříve hledají kódová slova v redukováném kódu  $\mathcal{K}'_S$ , který vznikne výběrem náhodnou množinou dimenzí  $S$  z matice  $\hat{G}'$ . Poté je nutné tato kódová slova rozšířit do původního kódu  $\mathcal{K}'$  a zkontrolovat, zda mají požadovanou *váhu*.

Algoritmy představené autory *Leon* [26], *Stern* [40] a *Canteaut* a *Chabaud* [11] se liší hlavně ve způsobu výběru dimenzí  $S$ . Poslední z představených algoritmů dosahuje nejlepších výsledků.

#### 4.1.5 Další útoky

Existují též návrhy dalších útoků jako jsou například statistické útoky [21] či útok založený na *bodových mřížích* [10]. Jako další zdroje pro zkoumání těchto útoků jsou doporučeny články [36, 15].

## 4.2 Bezpečné parametry

Pro dosažení určité míry bezpečnosti se používá pojem *počet bitů bezpečnosti* (či *míra bezpečnosti*). Tato jednotka odpovídá počtu bitů klíče symetrické šifry, které by útočník musel hrubou silou prolomit. Jinými slovy, pokud nějaká šifra (s danou velikostí klíče) odpovídá  $n$  bitům bezpečnosti, je třeba vynaložit  $O(2^n)$  operací.

Obecně je považováno *128 bitů bezpečnosti* za dostatečné pro *střednědobé* a *256 bitů* pro *dlouhodobé* účely. Méně než *80 bitů* je pro bezpečné uchování informací prakticky nepoužitelné, jelikož takto „silný“ algoritmus lze (či půjde) prolomit v dostatečně krátkém čase (méně než desítky let) [32].

Kryptosystém *McEliece* má na rozdíl např. od *RSA* několik parametrů –  $n$ ,  $k$ ,  $t$ ,  $\dots$  – a množství variant je tedy velmi široké. Odhady složitostí jednotlivých útoků se navíc celkem liší, a proto v této kapitole uvádíme několik tabulek z různých zdrojů, které odhadují *míru bezpečnosti* kryptosystému *McEliece*.

Tabulka 4.1 shrnuje parametry kryptosystému *McEliece* pro dosažení požadované míry bezpečnosti dle [6] a tabulka 4.2 dle [36]. Tyto tabulky obsahují informaci o velikosti *veřejného klíče* v *systematické formě*. Tabulka 4.3 inspirovaná z [15, 32] porovnává asymptotické složitosti šifrování a dešifrování kryptosystému *McEliece* a *RSA*.

Z těchto tabulek je vidět, že pro dosažení rozumné míry bezpečnosti je velikost veřejného klíče v řádech stovek *kilobajtů* až jednotek *megabajtů*. Ač je v dnešní době disková kapacita prakticky neomezená, tak hlavní problém je v přenosu tohoto klíče při navazování komunikace. Při běžné komunikaci se nejdříve (bezpečným) způsobem vyměňují klíče. V této fázi se při použití

Míra bezpečnosti	Parametry $(n, k, t)$	Velikost klíče
80 b	(1632, 1269, 33)	450 kb
128 b	(2960, 2288, 56)	1502 kb
256 b	(6624, 5129, 115)	7488 kb

Tabulka 4.1: Míra bezpečnosti *McEliece* dle [6]

Míra bezpečnosti	Parametry $(n, k, t)$	Velikost klíče
50 b	(1024, 524, 50)	256 kb
80 b	(2048, 1696, 32)	583 kb
128 b	(3178, 2384, 68)	1849 kb
128 b	(4096, 3604, 41)	1732 kb
256 b	(6944, 5208, 136)	8829 kb

Tabulka 4.2: Míra bezpečnosti *McEliece* dle [36]

Kryptosystém	Parametry	Míra bezpečnosti	Velikost klíče	Složitost	
				šifr.	dešifr.
<i>RSA</i>	1024b modul	$\sim 80$ b	1 kb	$2^{30}$	$2^{30}$
	2048b modul	$\sim 112$ b	2 kb	$2^{33}$	$2^{33}$
	4096b modul	$\sim 145$ b	4 kb	$2^{36}$	$2^{36}$
<i>McEliece</i>	(2048, 1608, 40)	$\sim 98$ b	691 kb	$2^{20}$	$2^{23}$
	(2048, 1278, 70)	$\sim 110$ b	961 kb	$2^{20}$	$2^{24}$
	(4096, 2056, 170)	$\sim 184$ b	4096 kb	$2^{22}$	$2^{26}$

Tabulka 4.3: Porovnání *McEliece* a *RSA* dle [15, 32]

*McEliece* musí vyměnit klíče často větší než poté samotná data. Navíc v porovnání s *RSA* – používanou asymetrickou šifrou s jinak řádově největšími klíči – je vidět, že tato čísla jsou prakticky nepřijatelná.

Na druhou stranu je vidět, že časová složitost algoritmů pro šifrování a dešifrování je naopak řádově menší. Kryptosystém *McEliece* má tedy v porovnání s *RSA* dobré vlastnosti co se týče propustnosti šifrování i dešifrování dat.

*Míra bezpečnosti* původního navrženého kryptosystému (1024, 524, 50) se dle [11, 36] pohybuje mezi 50-60 *bity bezpečnosti* a tyto parametry jsou tedy pro praktické použití nedostatečné.

### 4.3 Slabiny kryptosystému

V této kapitole shrneme známé slabiny kryptosystému *McEliece*, se kterými je nutné počítat a praktické použití šifrování pomocí *McEliece* náležitě upravit. Většina z těchto slabin umožňuje útok pomocí (adaptivně) voleného šifrovaného textu – tzv. *CCA2* útok,

Těmto slabinám se dá vyhnout díky použití *CCA2* bezpečné konverzi šifrovaného textu, kterou popíšeme v kapitole 5.2.

#### 4.3.1 Malleability

Použití šifrování tak, jak je definováno v kapitole 1.2 umožňuje deterministic-  
kým způsobem změnit (neznámou) zašifrovanou zprávu – tzv. *malleability*.

Zašifrovanou zprávu  $c_1$  veřejným klíčem  $\hat{G}$  jsme zkonstruovali (dle definice)  $c_1 = m_1\hat{G} + z$ , kde  $z$  je náhodný chybový vektor. Pokud tuto zprávu  $c_1$  zachytí útočník, může ji pozměnit následujícím způsobem:

- Připraví (otevřená) zpráva  $m_1$
- Tuto zprávu „zašifruje“ veřejným klíčem  $\hat{G}$ , ale nepoužije se chybový vektor  $z$ :  $c_2 = m_2\hat{G}$
- K původní zašifrované zprávě  $c_1$  přičte novou zprávu  $c_2$ :  $c = c_1 + c_2$
- Odešle vzniklou zprávu  $c$  původnímu účastníkovi.

Dešifrování proběhne naprosto bezchybným způsobem, ale účastník získá místo původní zprávy  $m_1$  podvrženou zprávu  $m_1 + m_2$ .

$$\begin{aligned} D_{SK}(c) &= D_{SK}(c_1 + c_2) = \\ &= D_{SK}\left((m_1\hat{G} + z) + m_2\hat{G}\right) = \\ &= D_{SK}\left((m_1 + m_2)\hat{G} + z\right) = \\ &= (m_1 + m_2) \end{aligned}$$

Podobnou slabinu mají i algoritmy *RSA* či *ElGamal* [46]. Stejně jako u těchto algoritmů (např. *OAEP* pro *RSA*) i pro *McEliece* se dá tomuto útoku efektivně bránit předem daným formátem zprávy a *paddingem*.

#### 4.3.2 Opakované šifrování stejné zprávy

Pokud je jedna otevřená zpráva dvakrát zašifrovaná stejným klíčem, je možné ji s velkou pravděpodobností odhalit [9]. Pro každé šifrování je generován náhodný (a pravděpodobně tedy jiný) chybový vektor  $z$ . Sečtením dvou různých

šifrových textů jedné zprávy tak získáme součet náhodných chybových vektorů:

$$c_1 + c_2 = (m\hat{G} + z_1) + (m\hat{G} + z_2) = z_1 + z_2$$

Váha každého z vektorů je  $t$  a délka  $n$ . Sečtením dvou šifrových textů tak získáme vektor váhy maximálně  $2t$ . Tento výsledný vektor pak obsahuje binární 1 na pozicích, kde se vyskytují 1 právě v jednom z chybových vektorů. Jelikož jsou chybové vektory velmi řídké, je velmi pravděpodobné, že výsledný vektor bude mít váhu právě  $2t$ . Pokud by vektory  $z_1$  a  $z_2$  obsahovaly 1 na stejných pozicích, váha výsledného vektoru by byla o 2 menší za každou takovou shodu. Počet možností chybového vektoru  $z_1$  je pak řádově nižší –  $\binom{2t}{t}$  místo původních  $\binom{n}{t}$ <sup>13</sup> – a útok s *informační množinou* je tak řádově jednodušší.

Dle stejného principu stačí znát *rozdíl* mezi dvěma zprávami. Označme tento rozdíl jako  $\Delta m = m_1 + m_2$ . Sečtením dvou odpovídajících šifrových textů získáme:

$$c_1 + c_2 = (m_1\hat{G} + z_1) + (m_2\hat{G} + z_2) = \Delta m\hat{G} + z_1 + z_2$$

Ze znalosti  $\Delta m$  a veřejného klíče je možné opět získat součet chybových vektorů  $z_1 + z_2$  a provést stejný útok na obě zprávy  $m_1$  a  $m_2$ , jak bylo uvedeno výše.

### 4.3.3 Znalost části otevřeného textu

Složitost útoku na šifrovanou zprávu lze též velmi zjednodušit, pokud útočník bude znát alespoň část otevřeného textu. Nechť množina  $\mathcal{I} \subset \{1, 2, \dots, k\}$  reprezentuje pozici bitů, které útočník zná. Potom  $\mathcal{J}$  je doplněk této množiny  $\mathcal{I}$  a zašifrovanou zprávu  $c$  lze rozdělit (dle dimenzí):

$$c = m\hat{G} + z = m_{\mathcal{I}}\hat{G}_{\mathcal{I}} + m_{\mathcal{J}}\hat{G}_{\mathcal{J}} + z$$

a tedy:

$$c + m_{\mathcal{I}}\hat{G} = m_{\mathcal{J}}\hat{G}_{\mathcal{J}} + z$$

$$\bar{c} = m_{\mathcal{J}}\hat{G}_{\mathcal{J}} + z$$

respektive:

$$\bar{c} = m_{\mathcal{J}}\hat{G}_{\mathcal{J}} + z_{\mathcal{J}}$$

Stačí tedy útočit na dimenze určené množinou  $\mathcal{J}$  a velikost *informační množiny* je tak zkrácena z  $k$  na velikost množiny  $\mathcal{J}$ .

<sup>13</sup> Pro praktické parametry kryptosystému platí  $n \gg t$ .

#### 4.3.4 Hádání chybových bitů

Tento útok je též označován jako tzv. „reakční útok“. Pro provedení tohoto útoku je třeba mít k dispozici *dešifrovací orákulum* a útočník musí být schopen rozlišit kdy došlo k chybě v dešifrování a kdy byla zpráva v pořádku dešifrována<sup>14</sup>.

Útočník, který zachytí zašifrovanou zprávu  $c$ , k ní přičte vektor s *Hammingovou vahou* 1:  $(0 \dots 010 \dots 0)$ . Takto upravenou zprávu odešle *orákulu* a pozoruje, jestli došlo k úspěšnému dešifrování či nikoliv. Pokud dešifrování selhalo, je jasné, že odeslaná upravená zpráva obsahovala  $t + 1$  chyb a nebylo možné přijatou zprávu dekodovat. Pokud dešifrování proběhne v pořádku, upravená zpráva obsahovala  $\leq t$  chyb, což znamená, že vektor, kterým byla zpráva upravena, odpovídá jednomu z náhodných bitů chybového vektoru  $z$ .

Útočník tímto způsobem může bit po bitu vyzkoušet úspěšnost dešifrování upravené zprávy a zrekonstruovat chybový vektor  $z$  v  $O(n)$  krocích. Za znalosti chybového vektoru je pak odhalení tajné zprávy  $m$  otázka vyřešení soustavy  $k$  rovnic v  $O(k^3)$  krocích.

Jako účinné zabránění tohoto útoku se nabízí vyžadovat, aby zašifrovaná zpráva obsahovala *právě*  $t$  chyb. Při šifrování se to dá velmi snadno zařídit a při dešifrování pak stačí zkontrolovat *váhu* chybového vektoru (který je získán při dekodování) a pokud není rovna  $t$ , je jasné, že došlo k manipulaci se šifrovým textem.

---

<sup>14</sup> Podobně jako např. útok *Padding Oracle* u blokových šifer [46].

## Moderní varianty a úpravy

Použití kryptosystému *McEliece* tak, jak byl popsán v 1. kapitole by bylo pro účely šifrování velmi nerozumné a nepraktické. To hlavně z důvodu slabin, kterými algoritmus trpí (viz kapitola 4.3) a velikosti klíčů, které jsou v základní variantě větší než je nezbytně nutné. V následujících kapitolách probereme několik úprav *kryptosystému* pro jeho praktické použití.

### 5.1 Metody na snížení velikosti klíčů

Jednou z hlavních nevýhod kryptosystému *McEliece* jsou obrovské klíče, které reprezentují lineární kódy velkých rozměrů (*Goppa kódy*) a matice odpovídající velikosti, které mají za úkol schovat strukturu použitého kódu. Metody na snížení velikosti klíčů se zaměřují hlavně na použití kódů, které je možné definovat kompaktním způsobem a způsob uložení či generování matic  $S$  a  $P$ .

Zatím byly všechny pokusy nahradit původní *Goppa kódy* jinými, kompaktnějšími lineárními kódy, neúspěšné. Nalezly se slabiny ve struktuře kódu, které lze využít pro jejich sestavení bez znalosti tajných matic  $S$  a  $P$  (viz kapitola 4.1.1. Jediné alternativní kódy, jejichž použití zatím nebylo prolomeno, jsou *kvazi-dyadické Goppa kódy*, které zmíníme v kapitole 5.1.3 a *MDPC kódy* v kapitole 5.1.4.

Kromě definovaného kódu jsou v *soukromém klíči* obsažené též dvě velké matice  $S$  a  $P$ . Snížením velikosti těchto matic se zabýváme v podkapitole .

#### 5.1.1 Matice $\hat{G}$ v systematické formě

Veřejný klíč je pouze jedna matice – „zamaskovaná“ generující  $k \times n$  matice  $\hat{G}$ . Jako jediný způsob pro snížení počtu bitů tohoto veřejného klíče je uložení matice v *systematické formě*. V takovém případě není třeba udávat prvních  $k$  sloupců – je jasné, že odpovídají *jednotkovým maticím*  $\mathbb{I}_k$ . Při použití matice  $\hat{G}$  v *systematické formě* se tedy ušetří  $k^2$  bitů, což při rozumných parametrech odpovídá až 75 % velikosti matice  $\hat{G}$ . Aby byla zachována bezpečnost kryp-

tosystému při použití takové matice, je nutné použít *CCA2 odolnou* konverzi (viz kapitola 5.2).

### 5.1.2 Význam matic $S$ a $P$

Jak jsme již zmínili v kapitole 1.3.2, *permutační* matici  $P$  není nutné ukládat jako matici bitů, ale pouze jako *indexy* permutace a velikost klíče tak komprimovat. Matice  $S$  je náhodná regulární matice a z definice nejde nijak komprimovat. Při hardwarové implementaci v [33] bylo ale efektivně využito *CSPRNG* jako *generátoru* této matice. Jednoznačnost matice  $S$  je zde vyjádřena pomocí tajného *seedu* pro *CSPRNG*.

Ač byl *kryptosystém* navržený s maticemi  $S$  a  $P$  pro *ukrytí generující* matice  $G$ , tak v [15] bylo ukázáno, že matice  $S$  nemá žádný bezpečnostní účel pro skrytí matice  $G$ . Naopak matice  $P$  je velmi důležitá a prozrazení této permutace by znamenalo prozrazení *soukromého klíče*.

### 5.1.3 Kvazi-dyadické Goppa kódy

Jako jedna z úspěšných metod na zkrácení klíčů se v posledních letech jeví použití *kvazi-dyadických (QD) Goppa kódů* [29].

**Definice 2** *Dyadická matice:*

- Každá  $1 \times 1$  matice je dyadická.
- Nechť  $A$  a  $B$  jsou  $2^{k-1} \times 2^{k-1}$  dyadické matice, pak  $2^k \times 2^k$  matice

$$H = \begin{pmatrix} A & B \\ B & A \end{pmatrix}$$

je také dyadická.

**Definice 3** *Kvazi-dyadická matice:* Matice, která není dyadická, ale skládá se z dyadických *submatic* je kvazi-dyadická.

Dyadickou matici  $H$  lze jednoznačně vyjádřit pomocí jediného (prvního) řádku matice. Z definice lze zkonstruovat celou původní matici  $H$ . Kvazi-dyadická matice lze tak vyjádřit pomocí prvních řádků *dyadických* submatic.

V [29] autoři ukázali, že je možné sestrojít (binární) *Goppa kód*, který má kontrolní matici v *dyadické* formě – tzv. *dyadický Goppa kód*. Takto sestrojený kód by bylo velmi snadné zrekonstruovat z veřejného klíče a navrhli tak použití *kvazi-dyadického Goppa kódu* – s kontrolní maticí v *kvazi-dyadické* formě.

S použitím *kvazi-dyadických Goppa kódů* je dosaženo  $n$  krát menších klíčů než za použití obecných (binárních) *Goppa kódů* [29]. Implementaci *kryptosystému* s *kvazi-dyadickými Goppa kódy* lze nalézt např. v [33, 24]



### 5.1.4 MDPC McEliece

Jedna z nejnovějších variant kryptosystému *McEliece* je použití *Moderate Density Parity-Check (MDPC)* kódů a kvazi-cyklických *MDPC* kódů. Autoři v [30] navrhli použití těchto kódů v roce 2013 a dokázali nalézt klíče o velikosti přibližně 4 kb (!), které odpovídají 80 bitům bezpečnosti.

## 5.2 CCA2 odolná konverze

V kapitole 4.3 jsme se zmínili, že základní varianta algoritmu *McEliece* trpí některými slabiny. Kvůli těmto slabinám by nebylo možné algoritmu prakticky (a opakovaně) využívat. Z tohoto důvodu bylo navrženo několik *konverzí*, které jsou odolné vůči útoku s *adaptivně voleným šifrovým textem* – CCA2 odolné konverze.

Jsou známé *obecné* konverze pro asymetrické šifry odolné vůči útoku s *voleným šifrovým textem* (CCA1). Například známá a používaná konverze *OAEP* v kryptosystému *RSA*. Dále to jsou například konverze *Fujisaki-Okamoto* a *Pointcheval*.

Nicméně *K. Kobara* a *H. Imai* v [22] uvádí, že tyto konverze nejsou CCA2-odolné a stále tak umožňují např. *reakční útok* (viz kapitola 4.3). Sami pak navrhli 3 možné CCA2-odolné konverze, z nichž třetí – označená jako *Kobara-Imai  $\gamma$  konverze* – je nejúčinnější. Tato konverze  $\gamma$  je popsána algoritmem 1 a ilustrována obrázkem 5.1. Algoritmus obsahuje v komentářích číselné odkazy do zmíněného obrázku.

### Značení

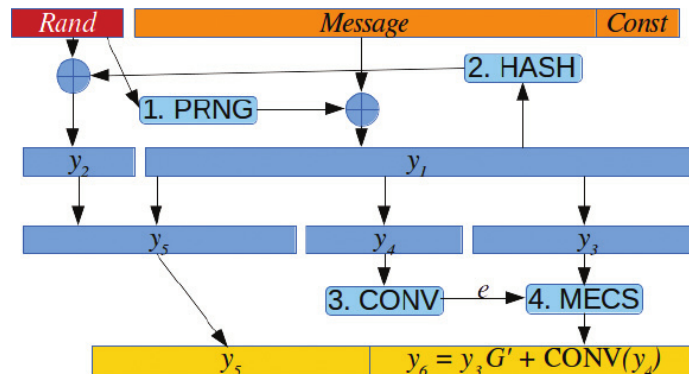
V následujícím algoritmu použijeme značení uvedené v tabulce 5.2 a délky vektorů v tabulce 5.1.

Vektor	Délka
$y_1$	$\max( rand , n +  const )$
$y_2$	$\max( r ,  hash )$
$y_3$	$k$
$y_4$	$\log_2 \binom{n}{t}$
$y_5$	$n +  const  +  r  -  y_4  -  y_3 $

Tabulka 5.1: Délky vektorů v algoritmu Kobara-Imai  $\gamma$

Symbol	Význam
$(a b)$	zřetězení vektorů $a$ a $b$
$m$	otevřený text
$const$	veřejně známá konstanta
$r$	náhodné číslo ( <i>seed</i> )
$prep(m)$	funkce na doplnění zprávy na požadovanou délku (jednoznačný <i>padding</i> )
$hash(l)$	kryptograficky bezpečná <i>hashovací</i> funkce s výstupem délky $\log_2 \binom{n}{t}$ bitů
$rand(r)$	kryptograficky bezpečná funkce inicializovaná <i>seedem</i> $r$ , která vrací (pseudonáhodný) vektor ( <i>CSPRNG</i> )
$conv$	invertibilní konverze čísla $\leq \binom{n}{t}$ na odpovídající vektor délky $n$ a váhy $t$ (viz též kapitola 2.2)
$E_{\hat{G}}(m, e)$	šifrovací algoritmus <i>McEliece</i> (vstupem je zpráva $m$ a chybový vektor $z$ )
$D_{SK}(c)$	dešifrovací algoritmus <i>McEliece</i> se soukromým klíčem $SK$
$MSB_n(l)$	$n$ nejvíce významných (levých) bitů vektoru $l$
$LSB_n(l)$	$n$ nejméně významných (pravých) bitů vektoru $l$

Tabulka 5.2: Použité značení v algoritmu Kobara-Imai  $\gamma$



Obrázek 5.1: Diagram CCA2 odolné konverze *Kobara-Imai*  $\gamma$  [36, 22]

**Algoritmus 1** Konverze Kobara-Imai  $\gamma$ 


---

```

1: function ENCRYPT(  $m, r, const$  )
2:    $\bar{m} \leftarrow \text{prep}(m)$ 
3:    $y_1 \leftarrow \text{rand}(r) + (\bar{m}|const)$  ▷ 1.
4:    $y_2 \leftarrow r + \text{hash}(y_1)$  ▷ 2.
5:    $(y_5|y_4|y_3) \leftarrow (y_2|y_1)$ 
6:    $e \leftarrow \text{conv}(y_4)$  ▷ 3.
7:    $c \leftarrow (y_5|E_{\hat{G}}(y_3, e))$  ▷ 4.
8:   return  $c$ 
9: end function

1: function DECRYPT(  $c, const$  )
2:    $y_5 \leftarrow \text{MSB}_{|c|-n}$ 
3:    $(y_3, e) \leftarrow D_{SK}(LSB_n(c))$  ▷ 4.
4:    $y_4 \leftarrow \text{conv}^{-1}(e)$  ▷ 3.
5:    $(y_2|y_1) \leftarrow (y_5|y_4|y_3)$ 
6:    $r \leftarrow y_2 + \text{hash}(y_1)$  ▷ 2.
7:    $(\bar{m}|const) \leftarrow y_1 + \text{rand}(r)$  ▷ 1.
8:   if  $const = const$  then
9:     return  $\text{prep}^{-1}(\bar{m})$ 
10:  else
11:    return  $NULL$  ▷ zamítni  $c$ 
12:  end if
13: end function

```

---



## Implementace

Pro implementaci kryptosystému *McEliece* v této práci jsme zvolili software *Wolfram Mathematica* [52]. Tento software jsme zvolili hlavně díky pohodlnosti některých matematických výpočtů a konstrukcí a také pro přehlednost výstupů.

Při implementaci *kryptosystému* se ukázaly nedostatky softwaru *Mathematica* a bylo nutné zpracovat problematiku (rozšířených) *konečných těles* a *binárních Goppa kódů*. Tyto dvě oblasti byly implementovány přímo v softwaru *Mathematica* tak, aby bylo možné jejich pohodlné použití i v jiných oblastech.

Celkově byla práce rozdělena do třech ucelených částí – (binární) *konečná tělesa*, (ireducibilní) *binární Goppa kódy* a *kryptosystém McEliece*. Každou z těchto částí lze využít jako *balík* či *knihovnu* pro další výpočty. V následujících sekcích popíšeme jednotlivé části implementace, včetně použitých algoritmů a příkladů výpočtů. Příklady použití a zdrojové kódy implementace jsou k nalezení na přiloženém disku a též online na <https://github.com/VojtechMyslivec/mceliece-mathematica>.

### 6.1 Binární konečná tělesa

V této podkapitole pojednáváme o implementaci *binárních konečných těles* včetně jejich *rozšíření*. V podkapitole 6.1.1 zmíníme existující řešení v softwaru *Mathematica* a v 6.1.2 odůvodníme a představíme zvolené řešení implementace. Dále se v podkapitole 6.1.3 detailně věnujeme implementovanými algoritmy a též jsou zde uvedené příklady výpočtů. Nakonec v 6.1.4 zmíníme některá možná zlepšení implementovaných operací.

**Poznámka:** Potřebná teorie z oblasti obecné algebry a především konečných těles je k nalezení v příloze A.

### 6.1.1 Existující řešení

Pro operace s *konečnými tělesy* v softwaru *Mathematica* byly prostudovány interní funkce pro operace s polynomy a externí balík `FiniteFields`. Vlastnosti těchto řešení popíšeme v následujících kapitolách.

#### 6.1.1.1 Operace s polynomy

Software *Mathematica* obsahuje funkce pro operace s polynomy nad reálnými (případně i komplexními) čísly. Většina těchto funkcí má volitelnou *možnost*<sup>15</sup> *Modulus*, díky které lze zajistit, aby operace s koeficienty byly prováděny nad celými čísly *modulo* zadané číslo  $p$ . Tímto způsobem je možné implementovat operace nad tělesy  $GF(p^n)$ , nicméně je téměř nemožné tímto způsobem implementovat *rozšířená tělesa* – polynomy nad polynomy.

Pro použití těchto funkcí (např. `ExtendedPolynomialGCD`, je třeba polynomu v úplném tvaru  $\sum a_i x^i$  – včetně  $x^i$  s tím, že  $x$  musí být nedefinovaný *symbol*<sup>16</sup>. Tento požadavek je celkem nepraktický, protože definování této proměnné kdekoli v programu by vedlo k nemožnosti použití těchto funkcí. Navíc udržovat si prvky ve formě např.  $x^6 + x^3 + x + 1$  místo 1001011 není pohodlné. Další nevýhoda použití polynomů je, že software *Mathematica* vypisuje polynomy od *nejnižšího* členu po *nejvyšší* (např.  $1 + x^2 + x^4 + x^7$ ), což je obrácený zápis, než je v technické literatuře zvykem.

#### 6.1.1.2 Rozšiřující balík `FiniteFields`

*Balík* v softwaru *Mathematica* je soubor obsahující rozšiřující funkce, které standardně nejsou k dispozici. Balík je možné načíst pomocí funkcí `Needs`, či případně `Get`.

Balík `FiniteFields` obsahuje základní operace pro práci s tělesy  $GF(p^n)$ . Prvky konečných těles jsou pak určeny *seznamem*<sup>17</sup> koeficientů a *hlavičkou*, která určuje do jakého tělesa prvek patří. Výhoda tohoto opatření je, že pro sčítání a násobení je pak možné využít obyčejné symboly operací (+, −, \*, /) a operace se automaticky provede v daném tělese. Pro parametry  $p$  a  $n$  je určené jedno těleso  $GF(p^n)$  (s jedním konkrétním ireducibilním polynomem) a *seznam* koeficientů prvku se opět píše od nejnižšího řádu po nejvyšší (například polynom  $x^3 + x + 1$  z tělesa  $GF(2^5)$  je zapsán jako  $GF[2, 5][\{1, 1, 0, 1, 0\}]$ .

Funkce z balíku `FiniteFields` nejsou dostatečně zdokumentovány, jak je jinak v softwaru *Mathematica* zvykem. Nepodařilo se využít funkcí z tohoto balíku pro operace s *rozšířenými tělesy*.

---

<sup>15</sup> Anglicky se tento termín v softwaru *Mathematica* nazývá *Option*.

<sup>16</sup> Jinými slovy proměnná, která nemá definovanou hodnotu.

<sup>17</sup> *Seznamem* se myslí struktura v softwaru *Mathematica* – *List*.

### 6.1.2 Zvolené řešení

Existující řešení pro práci s *konečnými tělesy* se ukázala jako nedostačující. Jejich hlavní nevýhodou je nemožnost použití při výpočtech s *rozšířenými tělesy*. Proto bylo implementováno vlastní řešení pro práci s *konečnými tělesy*.

Při implementaci operací nad *konečnými tělesy* bylo dodržováno následující jednotné rozhraní:

- Prvky *konečných těles* reprezentujeme *seznamem* koeficientů od nejvyššího po nejnižší.  
U *rozšířených těles* jsou koeficienty opět prvky *konečných těles*.  
Například polynom  $x^3 + x + 1$  je reprezentován seznamem:  $\{1, 0, 1, 1\}$   
a polynom  $(y + 1)x^2 + (y)$  je reprezentován:  $\{\{1, 1\}, \{0, 0\}, \{1, 0\}\}$
- Prvek (seznam koeficientů) může být libovolně dlouhý. V případě potřeby se při výpočtu *redukuje* (ireducibilním) polynomem nebo dorovná *nulovými* koeficienty.
- Počet koeficientů vnitřních prvků (koeficientů) musí být vždy stejný.  
Například prvek  $\{\{0, 0\}, \{1\}, \{1, 0\}\}$  není platný.
- Jednotlivým funkcím je kromě operandů předáván též i *modul* skládající se z odpovídajících (ireducibilních) polynomů, včetně charakteristiky tělesa. Tento *modul* je definovaný následovně:
  - Pro tělesa  $GF(p^{n_1})$  je *modul* složen z (ireducibilního) polynomu  $i_1$  stupně  $n_1$  a dané charakteristiky  $p$ :  
 $modul_1 = \{i_1, p\}$
  - Pro rozšířená tělesa se *modul* skládá z odpovídajícího polynomu  $i_k$  stupně  $n_k$  nad tělesem  $GF(p^{n_1}) \cdots^{n_{k-1}}$  a *modulu vnitřního tělesa*:  
 $modul_k = \{i_k, modul_{k-1}\}$ .
- Všem funkcím se předávají nejdříve *operandy* a poté *modul*.  
Například pro prvky  $a, b \in GF(p^{\dots})$ ,  $m \in \mathbb{N}$  a odpovídající *modul*:  
 $krat[a, b, modul]$   
 $inverze[a, modul]$   
 $mocnina[a, e, modul]$   
 $\dots$
- Pro implementaci operací v tělesech  $GF(p^n)$  jsou použité vnitřní funkce softwaru *Mathematica* pro práci s *polynomy*. Implementované funkce pro tato tělesa tedy zpravidla obsahují převod ze *seznamu* čísel na *polynom*, zavolání vnitřní funkce pro *polynom* a převodu zpět na *seznam* koeficientů. Díky těmto vnitřním funkcím je docíleno rychlejšího výpočtu, než kdyby byla použita vlastní implementace nad *seznamy* celých čísel.

- Pro implementaci operací v *rozšířených tělesech* byly implementovány jednotlivé algoritmy operací (popsané níže), jelikož nebylo možné použít pro tyto operace vnitřní funkce softwaru *Mathematica*. Funkce nad *rozšířenými tělesy* zpravidla volají odpovídající funkce ve vnitřních tělesech (například násobení jednotlivých *koefficientů*).

Tato pravidla umožňují pohodlný, jednotný a *rekurzivní* přístup k jednotlivým prvkům a voláním funkcí (druhá složka *modulu* je *modul vnitřního tělesa*, prvky *polynomu* jsou opět *polynomy*, ...).

**Poznámka:** Ač jsou funkce implementované v co nejobecnějším pojetí, tak je kladen důraz na efektivnost výpočtů vzhledem k *binárním* tělesům – tedy k *tělesům* s charakteristikou 2. Pro *tělesa* s jinou charakteristikou není chování funkcí definováno.

### 6.1.3 Implementace operací

V následujících kapitolách je popsána implementace hlavních operací v *konečných tělesech* a použitých algoritmů. Pro další informace je doporučeno nahlédnout do zdrojového kódu v souboru `src/rozsirenaBinarniTelesa.m` a příkladů použití.

V níže uvedených pseudokódech se používá některých prvků ze syntaxe softwaru *Mathematica*:

Zápis	Význam
<code>foo[bar]</code>	Volání funkce <i>foo</i> s argumentem <i>bar</i>
<code>ham[[i]]</code>	<i>i</i> -tý prvek seznamu (pole) <i>ham</i>

Tabulka 6.1: Prvky syntaxe jazyka softwaru *Mathematica*

#### 6.1.3.1 Sčítání

Jelikož operace sčítání se v jakémkoliv *tělese* provádí po jednotlivých koefficientech *modulo p*, je tato funkce jediná volána místo celkového modulu pouze se zadanou charakteristikou *p*.

Pro *rozšířená tělesa* funkce rekurzivně volá stejnou operaci sčítání na jednotlivé koefficienty zadaných polynomů až na úroveň obyčejných jednorozměrných seznamů. Pro sčítání těchto prvků funkce používá obyčejné sčítání dvou seznamů modulo *p*.



**Algoritmus 2** Sčítání prvků

---

```

1: function PLUS[  $a, b, p$  ] ▷ Pro  $GF(q^n)$ ,  $q$  je  $p^{\dots}$ 
2:   for  $i \leftarrow 1 \dots \text{Length}[a]$  do ▷ Cyklus implementován funkcí Map
3:      $c[[i]] \leftarrow \text{plus}[a[[i]], b[[i]], p]$ 
4:   end for
5:   return  $c$ 
6: end function

```

---

**Poznámka:** U operací s prvky z tělesa  $GF(p^n)$  jsou prvky – *seznamy*, např.  $\{1, 0, 1, 1\}$  – převáděny na polynomy – např.  $x^3 + x + 1$  – a využívá se vnitřních funkcí softwaru *Mathematica* pro práci s polynomy. Z tohoto důvodu jsou uváděny algoritmy pouze pro *rozšířená tělesa*  $GF(q^n)$ , kde  $q$  je nějaká mocnina prvočísla.

**6.1.3.2 Redukce polynomu**

Redukce polynomu (neboli *modulo* polynom) se používá ve většině dalších funkcí. Tato funkce se volá se dvěma parametry – prvkem  $a$  a polynomm (*modulem*)  $m$ . Funkce vrátí zbytek polynomu  $a$  po dělení polynomm  $m$ .

Redukce polynomu pro *rozšířená tělesa* je inspirovaná *Comb metodou* z [28]. K původnímu prvku  $a$  se opakovaně přičítá (od nejvyššího řádu) patřičný násobek *polynomu*  $m$  tak, aby se daný koeficient  $a_i$  rovnal nule (viz příklad níže).

Pro  $GF(p^n)$  se používá interní funkce `PolynomialMod`

**Algoritmus 3** Redukce prvku v tělese s charakteristikou 2

---

```

1: function REDUKUJ[  $a, \{m, \text{modul}_{\text{vnitrni}}\}$  ]
2:    $l_a \leftarrow \text{stupen}[a] + 1$  ▷ Délka redukovaného polynomu
3:    $l_m \leftarrow \text{stupen}[m]$  ▷ Výsledná délka redukovaného polynomu
   // Převedení  $m$  na monický polynom
4:    $\text{koe}f \leftarrow \text{inverze}[m[[1]], \text{modul}_{\text{vnitrni}}]$  ▷ Inverze nejvyššího koeficientu
5:    $m \leftarrow \text{krat}[\text{koe}f, m, \text{modul}_{\text{vnitrni}}]$  ▷ Násobení skalárem

6:    $m \leftarrow \text{PadRight}[m, l_a - l_m]$  ▷ Natáhnutí polynomu na délku  $a$ 
7:   for  $i \leftarrow 1 \dots l_a - l_m$  do
8:      $s \leftarrow \text{krat}[a[[i]], m, \text{modul}_{\text{vnitrni}}]$  ▷ Skalární násobek
9:      $a \leftarrow \text{plus}[a, s, 2]$  ▷ Odečtení v binárním tělese
10:     $m \leftarrow \text{RotateRight}[m]$  ▷ Posunutí redukovaného polynomu
11:  end for

12:  return  $a$ 
13: end function

```

---

**Příklad** Redukce polynomu  $(10)x^5 + (10)x^4 + (01)$  polynomem  $(10)x^3 + (01)x^2 + (11)x + (10)$  (nad tělesem  $GF(2^2)$  s ireducibilním polynomem 111):

$$\begin{array}{r} (10)(10)(00)(00)(00)(01) \mod (10)(01)(11)(10) : \\ \underline{(10)(10)(00)(00)(00)(01)} \\ (10)(01)(11)(10)(00)(00) \\ \underline{(00)(11)(10)(01)(11)(00)} \\ (00)(00)(01)(11)(10)(01) \\ \underline{(00)(01)(00)} \end{array}$$

$$\Rightarrow |(10)(10)(00)(00)(00)(01)|_{(10)(01)(11)(10)} = (00)(01)(00)$$

### 6.1.3.3 Násobení

Výsledkem násobení dvou polynomů  $a$  a  $b$  stupně  $n$  a  $m$  je polynom  $c$  stupně  $n + m$ . Násobení je implementováno tak, že k výsledku  $c$  (na počátku je to nulový polynom) se postupně přičítá skalární násobek polynomu  $b$  koeficienty polynomu  $a$ , který je zároveň *posunutý* o patřičný počet pozic. Využívá se zde faktu, že násobení libovolného *polynomu*  $a(x)$  a  $x^i$  je posunutí koeficientů polynomu  $A$  o  $i$  pozic doleva. Výsledný polynom  $c$  je následně *redukován* zadaným modulem (viz výše).

Pro  $GF(p^n)$  se používá obyčejného násobení dvou *polynomů* a následné *redukce modulem*.

---

#### Algoritmus 4 Násobení prvků

---

```

1: function KRAT[  $a, b, \{m, modul_{vnitrni}\}$  ]
2:    $p \leftarrow charakteristika[modul]$  ▷ Charakteristika tělesa
   // Natažení na výslednou délku
3:    $b \leftarrow PadLeft[b, stupen[a] + stupen[b] + 1]$ 
4:    $c \leftarrow nulovyPolynom[...]$  ▷ Nulový polynom nad vnitřním tělesem

5:   for  $i \leftarrow stupen \dots 1$  do
6:      $s \leftarrow krat[a[[i]], b, modul_{vnitrni}]$  ▷ Skalární násobek
7:      $c \leftarrow plus[c, s, p]$ 
8:      $b \leftarrow RotateLeft[b]$  ▷ Posunutí přičítaného polynomu
9:   end for

10:  return redukuj[ $c, modul$ ]
11: end function

```

---

**Příklad** Násobení polynomu  $(110)x^2 + (101)x + (001)$  polynomem  $(001)x^3 + (010)x + (011)$  (nad tělesem  $GF(2^3)$  s ireducibilním polynomem 1011):

$$\begin{array}{r}
 (110)(101)(001) \cdot (001)(000)(010)(011) : \\
 (001)x^3 : (110) (101) (001) (000) (000) (000) \\
 (000)x^2 : (000) (000) (000) (000) (000) (000) \\
 (010)x^1 : (000) (000) (111) (001) (010) (000) \\
 (011)x^0 : (000) (000) (000) (001) (100) (011) \\
 \hline
 (110) (101) (110) (000) (110) (011)
 \end{array}$$

$\Rightarrow$  Výsledek operace násobení modulo polynom  $g$  se získá redukcí polynomu  $(110)(101)(110)(000)(110)(011)$  polynomem  $g$ .

#### 6.1.3.4 Inverze

Výpočet multiplikativní *inverze* je implementován pomocí *rozšířeného Euklidova algoritmu*. Tento algoritmus se často vizualizuje jako výpočet tabulky po řádkách (viz níže). Ve skutečnosti však pro výpočet dalšího řádku stačí pracovat s hodnotami dvou řádků předešlých. Proto si není nutné udržovat v paměti celou tabulku, ale stačí si udržovat hodnoty dvou řádků a po výpočtu třetího hodnoty posunout.

Výpočet hodnot dalšího řádku tabulky probíhá následovně:

- Hodnoty předchozích řádků jsou:  
Polynomy  $p_{i-2}$  a  $p_{i-1}$  (na začátku inicializovány na ireducibilní polynom  $m$  a *prvek*, ke kterému je hledaná inverze).  
Polynomy  $k_{i-2}$  a  $k_{i-1}$  (na začátku inicializovány na 0 a 1, respektive *nulový* a *jednotkový polynom*).
- Je spočítán *podíl*  $q$  a zbytek  $p_i$  pomocí tzv. *dlouhého dělení* polynomu  $p_{i-2}$  polynomem  $p_{i-1}$ .
- Je spočítán *polynom*  $k_i = k_{i-2} - q \cdot k_{i-1}$
- Tyto kroky se opakují, dokud není získán polynom  $p_i$  stupně 0 (jinými slovy jediný prvek vnitřního tělesa).
- Výsledná *inverze* se získá jako skalární násobek *polynomu*  $k_i$  inverzí (posledního) *koefficientu* polynomu  $p_i$ .

Inverze v  $GF(p^n)$  je implementovaná pomocí interní funkce Polynomial-ExtendedGCD.

**Algoritmus 5** Inverze prvků – *Rozšířený Euklidův algoritmus*


---

```

1: function INVERZE[ prvek, modul : {m, modulvnitřní} ]
2:   A ← m; B ← prvek
   // Inicializace na jednotkový resp. nulový polynom z tělesa
3:   kA ← nulovyPolynom[...]; kB ← jednotkovyPolynom[...]
4:   while stufen[B] ≠ 0 do
   // Výpočet q a C pomocí dlouhého dělení v jednom kroku
5:     q ← A/B; C ← A mod B
6:     kC ← kA − krat[q, kB, modul]
7:     A ← B; kA ← kB
8:     B ← C; kB ← kC
9:   end while
   // Výpočet koeficientu ve vnitřním tělese
10:  koef ← inverze[Last[C], modulvnitřní]
11:  return krat[koef, kC, modulvnitřní]           ▷ Násobení skalárem
12: end function

```

---

**Poznámka:** Pro výpočet dělení je v *rozšířených tělesech* potřeba vypočítat inverzi největšího koeficientu dělitele<sup>18</sup> a dále je algoritmus realizován posouváním dělitele a následnou redukcí pomocí sčítání.

**Příklad** *Rozšířený Euklidův algoritmus* pro výpočet *inverze* polynomu  $(101)x^3 + (010)x^2 + (110)x + (111)$  modulo  $(001)x^4 + (011)x^3 + (011)x^2 + (001)x + (011)$  (nad tělesem  $GF(2^3)$  s ireducibilním polynomem 1101):

Podíl	Zbytek	Koeficient
	(001)(011)(011)(001)(011)	(000)
	(101)(010)(110)(111)	(001)
(111)(000)	(110)(011)(011)	(111)(000)
(111)(001)	(001)(100)	(010)(111)(001)
(110)(001)	(111) (001)(111)(110)(001)	

$$\Rightarrow |(101)(010)(110)(111)^{-1}|_{(001)(011)(011)(001)(011)} = (101)(001)(100)(101)$$

**6.1.3.5 Druhá mocnina**

Pro prvky tělesa s *charakteristikou* 2 je výhodné implementovat funkci „na druhou“ díky následujícímu tvrzení:

**Tvrzení 1** *Nechť*  $A = (a_n \dots a_2 a_1 a_0)$  *je prvek tělesa s charakteristikou* 2, *potom platí:*

$$A^2 = (a_n^2 0 \dots 0 a_2^2 0 a_1^2 0 a_0^2)$$

---

<sup>18</sup> Zde je patrná rekursivní vlastnost tohoto algoritmu, kdy pro výpočet inverze prvku v tělese  $GF(q^n)$  je třeba vypočítat inverzi v tělese  $GF(q)$ .

S využitím tohoto tvrzení je realizace funkce na počítání druhé mocniny triviální:

- Provedení druhé mocniny všech koeficientů.
- Proložení koeficientů polynomu nulovými koeficienty.
- Redukování polynomem (viz výše).

---

**Algoritmus 6** Umocňování na druhou v tělese s charakteristikou 2

---

```

1: function NADRUHOU[  $a, \{m, modul_{vnitrni}\}$  ]
2:   for  $i \leftarrow 1 \dots Length[i]$  do           ▷ Cyklus implementován funkcí Map
3:      $a[[i]] \leftarrow naDruhou[a[[i]], modul_{vnitrni}]$ 
4:   end for
5:    $nula \leftarrow nulovyPolynom[\dots]$            ▷ Odpovídající nulový koeficient
6:    $a \leftarrow Rif fle[a, nula]$                  ▷ Proloží koeficienty prvkem  $nula$ 
7:   return  $reduku jPolynom[a, modul]$ 
8: end function

```

---

**Náznak důkazu**

$$\begin{aligned}
 a(x) &= a_n x^n + \dots + a_2 x^2 + a_1 x + a_0 \\
 a(x)^2 &= (a_n x^n + \dots + a_2 x^2 + a_1 x + a_0) \cdot (a_n x^n + \dots + a_2 x^2 + a_1 x + a_0) = \\
 &= a_n x^n \cdot (a_n x^n + \dots + a_2 x^2 + a_1 x + a_0) + \\
 &\quad \vdots \\
 &+ a_2 x^2 \cdot (a_n x^n + \dots + a_2 x^2 + a_1 x + a_0) + \\
 &+ a_1 x \cdot (a_n x^n + \dots + a_2 x^2 + a_1 x + a_0) + \\
 &+ a_0 \cdot (a_n x^n + \dots + a_2 x^2 + a_1 x + a_0) = \\
 &= a_n^2 x^{2n} + \dots + a_n a_2 x^{n+2} + a_n a_1 x^{n+1} + a_n a_0 x^n + \\
 &\quad \vdots \\
 &+ a_n a_2 x^{n+2} + \dots + a_2^2 x^4 + a_2 a_1 x^3 + a_2 a_0 x^2 + \\
 &+ a_n a_1 x^{n+1} + \dots + a_2 a_1 x^3 + a_1^2 x^2 + a_1 a_0 x + \\
 &+ a_n a_0 x^n + \dots + a_2 a_0 x^2 + a_1 a_0 x + a_0^2 = \\
 &= a_n^2 x^{2n} + \dots + 2(a_3 a_0 + a_2 a_1) x^3 + 2(a_2 a_0) x^2 + a_1^2 x^2 + 2(a_1 a_0) x + a_0^2 = \\
 &= \sum_{i=0}^n a_i^2 x^{2i} + 2 \sum_{i=1}^{n+1} \sum_{\substack{j < k \\ j+k=i}} a_j a_k = \\
 &= \sum_{i=0}^n a_i^2 x^{2i} \cong (a_n^2 0 \dots 0 a_2^2 0 a_1^2 0 a_0^2)
 \end{aligned}$$

### 6.1.3.6 Obecné umocňování

Mocnění *polynomů* je implementováno pomocí algoritmu *Square-and-Multiply* (*SM*). Algoritmus využívá faktu, že libovolnou mocninu lze rozložit na součin mocnin čtverců ( $2^2, 4^2, 8^2, \dots$ ). Konkrétně byla implementována varianta provádějící výpočet od nejvíce významného bitu exponentu<sup>19</sup>. Algoritmus má vstupy polynom  $a$  a exponent  $e$ . Exponent se vyjádří jako číslo v *binární* soustavě a poté algoritmus provádí cyklus přes bity tohoto rozvoje. V každém kroku se mezivýsledek umocní na druhou a v případě, že je odpovídající bit exponentu 1, přinásobí se původní číslo  $a$ .

---

**Algoritmus 7** Umocňování prvku  $a^e \bmod \text{modul}$  – *Square-and-Multiply*

---

```
1: function UMOCHNI[  $a, e, \text{modul}$  ]
2:   if  $e = 0$  then
3:     return nulovyPolynom[...]           ▷ Nulový prvek tělesa
4:   end if
5:    $\text{rozvoj} \leftarrow \text{IntegerDigits}[e, 2]$    ▷ Binární rozvoj exponentu
6:    $c \leftarrow a$                            ▷  $\text{rozvoj}[[1]]$  je vždy 1
7:   for  $i \leftarrow 2 \dots \text{Length}[\text{rozvoj}]$  do
8:      $s \leftarrow \text{naDruhou}[c, \text{modul}]$ 
9:      $m \leftarrow \text{krat}[s, a, \text{modul}]$ 
10:    if  $\text{rozvoj}[[i]] = 1$  then
11:       $c \leftarrow s$ 
12:    else
13:       $c \leftarrow m$ 
14:    end if
15:  end for
16:  return  $c$ 
17: end function
```

---

**Poznámka:** Takto implementovaný algoritmus je zranitelný vůči odběrové a časové analýze. Pro odolnou implementaci je nutné počítat násobek *vždy* a pokud je daný bit exponentu 1, přiřadit násobek do mezi výpočtu. Pseudokód i reálná implementace je prováděna tímto (bezpečným) způsobem.

---

<sup>19</sup> Uváděna jako *MSB* – z anglického *Most Significant Bit*.

**Příklad** Algoritmus *Square-and-Multiply* pro výpočet  $((11)x^2 + (10))^{26} \bmod (01)x^3 + (11)x + (01)$  (nad tělesem  $GF(2^2)$  s ireducibilním polynomem 111):

Op.	Mocnina		Výpočet	Výsledek
	dek.	bin.		
	1	1		(11)(00)(10)
<b>S</b>	2	1	(10)(00)(00)(00)(11)	(01)(10)(11)
<b>M</b>	3	11	(01)(10)(11) · (11)(00)(10)	(10)(11)(00)
<b>S</b>	6	110	(11)(00)(10)(00)(00)	(11)(00)
<b>S</b>	12	1100	(10)(00)(00)	(10)(00)(00)
<b>M</b>	13	1101	(10)(00)(00) · (11)(01)(10)	(01)(00)
<b>S</b>	26	11010	(01)(00)(00)	(01)(00)(00)

$$\Rightarrow |(11)(00)(10)^{26}|_{(01)(00)(11)(01)} = (01)(00)(00)$$

#### 6.1.4 Možná zlepšení

V této kapitole nastíníme možná zlepšení implementace, která zrychlují výpočet některých operací.

##### Logaritmické tabulky

Pro zrychlení výpočtu násobení a mocnin prvku lze v *konečném tělese* využít faktu, že vždy existuje *primitivní prvek* a převádět tak operace v tělese na operace s celými čísly.

**Definice 4** *Nechť  $\alpha$  je generátor multiplikativní grupy tělesa  $\mathbb{F}$ . Potom říkáme, že  $\alpha$  je primitivní prvek tělesa  $\mathbb{F}$ .*

**Důsledek** Každý prvek tělesa  $\mathbb{F}$  – kromě *nulového* prvku *aditivní grupy* – lze vyjádřit jako  $\alpha^i$  pro nějaké  $i$ .

Důkaz plyne přímo z definice.

Násobení dvou prvků  $a = \alpha^{i_a}$  a  $b = \alpha^{i_b}$  tak můžeme převést na součet mocnin *primitivního prvku*:

$$a \cdot b = \alpha^{i_a} \cdot \alpha^{i_b} = \alpha^{i_a + i_b}$$

Podobným způsobem můžeme zjednodušit umocňování prvku:

$$a^e = \left(\alpha^i\right)^e = \alpha^{ie}$$

V obou případech je samozřejmě možné použít *Eulerovu větu* a mocniny redukovat *modulo*  $N$ , kde  $N$  je počet prvků *multiplikativní grupy tělesa* ( $N = p^n - 1$  pro těleso  $GF(p^n)$ ). Jakoukoliv operací násobení a mocnění získáme prvek  $\alpha^{n_c}$ , kde  $n_c$  je celé číslo v rozsahu od 0 do  $N - 1$ .

Reprezentací prvků pomocí odpovídajících mocnin *primitivního prvku* se tak můžeme vyhnout násobení a umocňování prvků v tělese a nahradit ho sčítáním a násobením celých čísel, což je řádově jednodušší. V případě sčítání prvků v tělese je však nutné mít jejich standardní reprezentaci (seznam koeficientů), jelikož se sčítání provádí po jednotlivých koeficientech, respektive bitech. Není možné nahradit sčítání dvou prvků jinou operací s mocninami *primitivního prvku*.

Pro použití tohoto zrychlení výpočtů je tak nutné připravit v paměti programu překladové *log-* a *antilogaritmičké* tabulky pro překlad prvků z jedné reprezentace na druhou.

Ač se tak získá podstatné zrychlení výpočtů v tělese, existuje několik nevýhod tohoto přístupu:

- Je nutné nalézt *primitivní prvek tělesa*.
- Je nutné vygenerovat a uchovat v paměti počítače obě tabulky pro překlad.
  - Tuto tabulku lze implementovat pomocí obyčejného pole či seznamu, kde se k danému indexu v seznamu vyskytuje odpovídající hodnota.
  - Pro binární tělesa  $GF(2^m)$  je velikost jedné tabulky  $O(m2^m)$  (konkrétně  $2^m - 1$  hodnot, kde každá je reprezentována  $m$  bity).
  - Jelikož je paměťová náročnost *exponenciální*, můžeme tyto tabulky uchovávat pouze pro *malá*  $m$  (např. 8 či 16, nikoliv však 1024).
- *Nulový prvek* tělesa není možné žádným způsobem zobrazit jako mocninu. Při každé operaci je potřeba s touto skutečností počítat a hlídat jako výjimku.

Toto vylepšení bychom mohli využít pro operace ve *vnitřním tělese*  $GF(2^m)$ , nad kterým jsou postavené polynomy v *binárních Goppa kódech*.

### Implementace dělení

Dělení prvkem  $b$  v *konečném tělese* převádíme na násobení  $b^{-1}$ . Pro výpočet *podílu* se tak počítá inverze a následně násobek. Je ale možné implementovat rovnou algoritmus pro dělení.

Algoritmus pro dělení prvku  $a$  prvkem  $b$  je totožný s algoritmem pro výpočet *inverze* prvku  $b$  s tím rozdílem, že je počáteční hodnota koeficientu  $k_b$  (viz EEA – alg. 5) nastavena na hodnotu  $a$ . Výsledkem algoritmu pak bude inverze prvku  $b$  vynásobená  $a$ , což přesně odpovídá výrazu  $a/b$ .



## 6.2 Ireducibilní binární Goppa kódy

Pro implementaci kryptosystému jsme zvolili *ireducibilní Goppa kódy*, které jsme popsali v kapitole 3. V podkapitole 6.2.1 popíšeme algoritmy, které slouží pro vygenerování kódu dle zadaných parametrů a v 6.2.2 *Pattersonův* algoritmus pro dekódování a (opravu) chyb vzniklých při přenosu. Zdrojový kód této části implementace je k nalezení v souboru `src/ireducibilni-BinarniGoppaKody.m`.

V podkapitole 6.2.3 uvedeme krátký příklad generování *Goppa kódu* a na konec v 6.2.4 zmíníme možná zlepšení implementovaných algoritmů.

### 6.2.1 Generování Goppa kódu

Parametry *ireducibilního binárního Goppa kódu*  $(n, k, t)$  jsou jednoznačně určeny parametry  $m$  a  $t$ . Parametr  $m$  určuje řád vnitřního tělesa  $GF(2^m)$  a parametr  $t$  stupeň (ireducibilního) *Goppova* polynomu  $g$ . Parametr  $n$  (počet prvků podpory  $L$ ) je  $2^m$ , protože posloupnost  $L$  obsahuje *všechny* prvky z tělesa  $GF(2^m)$ . Redundance takového kódu je  $r = mt$ , jelikož vygenerovaná matice  $H$  má  $t$  řádků nad tělesem  $GF(2^m)$ , neboli  $mt$  řádků nad tělesem  $GF(2)$ . Parametr  $k$  je pak jednoznačně určený z definice kódu jako  $n - k$  tedy  $2^m - mt$ .

Pro sestrojení *kontrolní* matice  $H$  potřebujeme vygenerovat *podporu* kódu  $L$  a matice  $V$  a  $D$ . Pro sestrojení těchto objektů je též třeba vygenerovat *modul* vnitřního tělesa  $GF(2^m)$  a samozřejmě *Goppův* polynom  $g$ . Pro sestrojení *modulu* využijeme funkcí definovaných v předešlé kapitole 6.1 a pro vygenerování matic a podpory jsou implementovány dílčí funkce popsané níže.

#### Generování Goppova polynomu

Pro sestrojení *ireducibilního Goppa kódu* je potřeba vygenerovat *ireducibilní* polynom nad konečným tělesem  $GF(2^m)$ . Pro skutečné generování náhodného ireducibilního polynomu by bylo třeba implementovat *test irreducibility polynomu* (např. dle [18]). Pro použití *Goppa* kódů v této práci bylo několik ireducibilních polynomů předgenerováno v softwaru *SageMath* [53].

#### Podpora $L$

Generování podpory  $L$  je velmi jednoduché. Dle parametru  $m$  se vygenerují všechny vektory z  $\{0, 1\}^m$  a náhodně se permutují (zamíchají). Tuto funkci lze jednoduše řešit pomocí vnitřních funkcí softwaru *Mathematica* `Tuple` a `RandomSample`.

#### Matice $D$

Matice  $D$  je diagonální  $n \times n$  maticí (nad  $GF(2^m)$ ), kde na diagonále jsou inverze v  $GF(2^m)$  prvků z  $L$  dosazených do polynomu  $g$ , neboli  $D_{i,i} = g(L_i)^{-1}$ . Pro výpočet této matice je potřeba zadat *modul* tělesa  $GF(2^m)$ , polynom  $g$

**Algoritmus 8** Generování Goppa kódu

---

```

1: function GENERUJGOPPAKOD[  $m, t$  ]
2:    $n \leftarrow 2^m; \quad r \leftarrow tm; \quad k \leftarrow n - r$ 
   // ireducibilní Goppův polynom  $g$  stupně  $t$  nad tělesem  $GF(2^m)$ 
3:    $modul \leftarrow generujModul[\{2, m\}, t]$ 
4:    $\{g, modul_{vnitrni}\} \leftarrow modul$ 

5:    $L \leftarrow generujPodporuL[m]$   $\triangleright$  Generování posloupnosti  $L$ 
6:    $V \leftarrow maticeV[podporaL, t, modul_{vnitrni}]$   $\triangleright$  Vandermondova matice
7:    $D \leftarrow maticeD[podporaL, modul]$   $\triangleright$  Diagonální matice

8:    $H \leftarrow dotNadF[V, D, modul_{vnitrni}]$   $\triangleright$  Násobení matic nad  $GF(2^m)$ 
9:    $H \leftarrow Flatten[Transpose /@ H, 1]$   $\triangleright$  Rozbalení prvků matice  $H$ 
   // Převod matice  $H$  na  $G$  – ortogonální doplněk
10:   $G \leftarrow NullSpace[H, Modulus \rightarrow 2]$ 

11:   $X \leftarrow \{jednotkovyPolynom[...], nulovyPolynom[...]\}$   $\triangleright$  Polynom  $x$ 
   // předpočítané dílčí syndromy  $(x - L_i)^{-1}$ 
12:  for  $i \leftarrow 1 \dots n$  do  $\triangleright$  Ve skutečnosti pomocí funkce Map
13:     $syndromyL[[i]] \leftarrow inverze[plus[X, L[[i]], 2], modul]$ 
14:  end for

15:  return  $\{G, modul, L, syndromyL\}$ 
16: end function

```

---

a podporu  $L$ . Výpočet je pak proveden pomocí *aplikování* (Map) funkcí dosažení do polynomu a inverze v tělese na prvky seznamu  $L$ .<sup>20</sup>

**Matice  $V$** 

Vandermondova  $t \times n$  matice  $V$  nad  $GF(2^m)$  obsahuje na prvním řádku *jednotkové prvky* a na dalších řádcích mocniny prvků z  $L$ . Konkrétně tedy  $V_{j,i} = L_i^{j-1}$ , pro  $i \geq 2$ . Vypočítání všech mocnin pro každé  $i, j$  by bylo velmi neefektivní. Rychlejší způsob je vygenerovat první řádek *jednotkových* prvků a každý další řádek vypočítat přináobením příslušného  $L_i$  k řádku předěšlému. Tento výpočet je realizován funkcí softwaru *Mathematica* NestList.

**6.2.2 Pattersonův algoritmus**

Pro zakódování zprávy do kódového slova stačí použít prostého maticového násobení (nad  $GF(2)$ ). Pro dekódování, respektive opravu chyb byl implementovaný *Pattersonův algoritmus*, který byl uveden v kapitole 3.2.

---

<sup>20</sup> Zde by šlo dosazení do polynomu všech prvků z  $L$  zrychlit stejným způsobem, jako byl uvedený u *Chienova* hledání kořenů v kapitole 3.2.

**Algoritmus 9** Dekódování Goppa kódu

---

```

1: function DEKODUJGOPPAKOD[  $c, G, \{g, modul_{vnitrni}\}, L, syndromyL$  ]
2:    $(n, k, t); m$  ▷ Parametry Goppa kódu – dle  $G$  a  $g$ 
   // Syndrom  $s(x) = \sum_{i=1}^n \frac{c_i}{x-L_i} \mod g(x)$ .
3:    $s \leftarrow \sum c[[i]] \cdot syndromyL[[i]]$  ▷ Realizováno funkcí Apply a Plus
4:   if  $s = 0$  then ▷ Pokud je syndrom nulový, chyba nenastala
5:      $e \leftarrow nulovyPolynom[2, n]$ 
6:   else ▷ Jinak provede opravu chyb – Pattersonův alg.
   // Polynom  $x$ 
7:      $X \leftarrow \{jednotkovyPolynom[...], nulovyPolynom[...]\}$ 
   //  $r = \sqrt{s(x)^{-1} - x} \mod g(x)$ 
8:      $r \leftarrow plus[inverze[s, modul], X, 2]$ 
9:      $r \leftarrow umocni[r, 2^{mt-1}, modul]$  ▷ Výpočet odmocniny
   // Rozložení polynomu  $r(x)$ :  $\alpha(x) = \beta(x)r(x) \mod g(x)$ 
10:     $\{\alpha, \beta\} \leftarrow modifikovanyEEA[r, modul]$ 

11:     $\beta \leftarrow posunPolynom[\beta^2, 1]; \alpha \leftarrow \alpha^2$ 
   // Lokátor chyb  $\sigma = \beta^2 x + \alpha^2$  – bez redukce  $g$ !
12:     $\sigma \leftarrow plus[\beta, \alpha, 2]$ 

13:    for  $i \in 1 \dots n$  do ▷ Dosazení  $L_i$  do lokátoru (opět pomocí Map)
14:       $e[[i]] \leftarrow dosadDoPolynomu[\sigma, L[[i]], modul_{vnitrni}] == 0$ 
15:    end for
16:  end if

17:   $c' \leftarrow plus[c, e, 2]$  ▷ Opravené přijaté slovo  $c$ 
   // Invertování zakódování maticí  $G$ 
18:   $d \leftarrow invertujZakodovani[c', G]$ 

19:  return  $\{d, e\}$  ▷ Vráti dekódovanou zprávu i chybový vektor
20: end function

```

---

**Druhá mocnina**

Druhá mocnina je implementována přímo v rámci algoritmu, jelikož je třeba vynechat redukci polynomu. Tato druhá mocnina se vypočítá<sup>21</sup> stejným způsobem, jako byl uveden v kapitole 6.1.3.5.

**Modifikovaný *EEA***

Rozložení polynomu  $r$  je realizováno modifikovaným *rozšířeným Euklidovým algoritmem*, jak bylo popsáno v kapitole 3.2.

---

<sup>21</sup> Výpočet druhé mocniny pomocí *kratBezRedukce* $[a, a, modul_{vnitrni}]$  není efektivní.

**Dosazení do polynomu**

Funkce `dosadDoPolynomu` je implementována v části zabývající se konečnými tělesy a výpočet dosazení prvku do polynomu je realizován pomocí tzv. *Hornerova schématu*.

**Invertování zakódování**

Posledním krokem algoritmu je získání původní zprávy  $d$  z opraveného. Bit zprávy  $d$  na  $i$ -té pozici odpovídá bitu vektoru  $c'$  na pozici sloupce matice  $G$ , který má v  $i$ -tém řádku 1 a v ostatních 0. V tomto kroku vlastně vyhledáme pozice *informačních bitů*, které tvoří původní zprávu  $d$ .<sup>22</sup>

**6.2.3 Ukázka**

V této sekci uvádíme krátkou ukázkou vygenerovaného kódu s parametry  $m = 3$  a  $t = 2$ . Ukázky s většími parametry a příklady použití jsou připravené v souboru `ireducibilniBinarniGoppaKody.nb`.

**Příklad:**

Máme ireducibilní *Goppův* polynom  $g(x) = (001)x^2 + (100)x + (001)$  nad tělesem  $GF(2^3)$  s ireducibilním polynomem 1011.

Vygenerujeme podporu  $L$ , což je posloupnost o  $2^m = 8$  prvcích z vnitřního tělesa  $GF(2^m)$ , neboli náhodná permutace všech prvků z tohoto tělesa.

$$L = (100, 001, 111, 011, 010, 000, 101, 110)$$

Sestrojíme *Vandermondovu* matici  $V$  a *diagonální* matici  $D$ . Pro prvky matice  $V$  platí předpis  $V_{j,i} = L_i^{j-1}$ , tedy první řádek jsou *jednotky* z *vnitřního* tělesa a po řádkách vždy přinásobíme *podporu*  $L$ .

$$V = \begin{pmatrix} 001 & 001 & 001 & 001 & 001 & 001 & 001 & 001 \\ 100 & 001 & 111 & 011 & 010 & 000 & 101 & 110 \end{pmatrix}$$

Prvky na hlavní diagonále matice  $D$  mají tvar  $D_{i,i} = g(L_i)^{-1}$ :

$$D = \begin{pmatrix} 001 & & & & & & & \\ & 111 & & & & & & \\ & & 110 & & & & & \\ & & & 110 & & & & \\ & & & & 011 & & & \\ & & & & & 001 & & \\ & & & & & & 111 & \\ & & & & & & & 011 \end{pmatrix}$$

---

<sup>22</sup> Jedná se vlastně o řešení soustavy  $k$  rovnic pro  $k$  neznámých určených vybranou maticí  $G_K$ . Je jistě nejjednodušší vybrat si takové dimenze  $K$ , že  $G_K$  je jednotková matice.

Vynásobením těchto dvou matic získáme *kontrolní* matici  $H$  (nad  $GF(2^m)$ ) *ireducibilního binárního Goppa kódu*  $\Gamma$ :

$$H = VD = \begin{pmatrix} 001 & 111 & 110 & 110 & 011 & 001 & 111 & 011 \\ 100 & 111 & 100 & 001 & 110 & 000 & 110 & 001 \end{pmatrix}$$

a rozbalením vektorů do sloupců nad  $GF(2)$ :

$$H = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}$$

$H$  je pak možné převést na generující matici  $G$  tohoto kódu (standardním způsobem pro *lineární kódy*, který je případně uveden v příloze B.2):

$$G = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 & 1 \end{pmatrix}$$

kde *informačními* bity jsou tedy například poslední dva bity přijatého slova.

Nakonec se předpočítají *syndromy* (posloupnost polynomů z  $GF(2^m)^t$ ) dle definice  $S_{L_i} = (x - L_i)^{-1} = ((001)(000) + L_i)^{-1} = ((001)(L_i))^{-1}$

$$S_L = ( \begin{array}{cccc} (001)(000), & (111)(110), & (110)(001), & (110)(100), \\ (011)(001), & (001)(100), & (111)(111), & (011)(110) \end{array} )$$

Kód  $\Gamma$  je určen (dle definice) *polynomem*  $g(x)$ , *podporou*  $L$  a případně vnitřním tělesem  $GF(2^m)$  (jeho modulem). K těmto hodnotám ještě přidáme předpočítané *syndromy*  $S_L$ , aby se při každém dekódování nebylo nutné zbytečně počítat. Tento vygenerovaný kód  $\Gamma$  má parametry  $(n, k, t) = (8, 2, 2)$

#### 6.2.4 Možná zlepšení

V následujících odstavcích se krátce zmíníme o více či méně významných zlepšeních, který by bylo vhodné implementovat.

##### Generování ireducibilních polynomů

Pro praktické použití Goppa kódů v oblasti bezpečnosti je potřeba generovat *ireducibilní* polynomy  $g$  náhodně. Pro účely této práce bylo předgenerováno pouze několik ireducibilních polynomů a v případě rozvíjení této implementace by bylo vhodné soustředit se primárně na tuto skutečnost.

**Hledání kořenů**

Nejnáročnější část dekodování je hledání kořenů *lokátoru chyb*  $\sigma$ . Opakované dosazování do polynomu lze efektivněji implementovat pomocí *Chienova* hledání kořenů, či případně faktorizací polynomu  $\sigma$ . Více informací ohledně tohoto problému jsme uvedli v kapitole 3.2.

**Invertování zakódování**

V případě, že je matice generována v *systematické* formě, tak zprávě  $d$  odpovídá prvních  $k$  (informačních) bitů vektoru  $c'$ . Matice  $G$  však v *systematické* formě nemusí existovat. Aby matici šlo sestavit v této formě, museli bychom prohazovat dimenze kódu a tím pádem i posloupnosti  $L$ . To není z důvodu bezpečnosti žádoucí, neboť bychom snižovali počet možných permutací podpory  $L$  a tím pádem i počet možných využitelných kódů. Z tohoto důvodu je nutné invertování zakódování maticí  $G$  provádět způsobem, který jsme popsali výše. Nicméně nalezení daných dimenzí není nutné provádět opakovaně a mohli bychom si tento krok předpočítat a v definici kódu uvádět definici pozic informačních bitů.

**Řád tělesa**

Stejně tak je při každém dekodování počítán počet prvků tělesa pro vypočítání odmocniny. Tento počet prvků – respektive číslo, na které je nutné prvek umocnit, abychom našli odmocninu – je možné pro zrychlení výpočtu též uložit mezi parametry definující kód. Oproti ostatním operacím se však jedná o minoritní výpočet a tak toto zrychlení by nebylo nijak významné.

## 6.3 McEliece

Pro ukázkovou implementaci kryptosystému *McEliece* jsme zvolili základní variantu *kryptosystému*, která byla popsána v 1. kapitole. Jedná se tedy o tzv. „školní variantu“, která by v praktickém použití selhala, a to hlavně kvůli slabínám, které jsme popsali v kapitole 4.3. Pro praktické použití by bylo třeba implementovat tzv. *CCA2* odolnou konverzi, kterou jsme popsali v kapitole 5.2). Jedná se však pouze o rozšíření základní varianty a je tedy možné tuto základní implementaci v budoucnu rozvinout tak, aby zmíněnou *CCA2* odolnou konverzi též obsahovala. Zdrojový kód této části implementace je k nalezení v souboru `src/mceliece.m`.

V podkapitole 6.3.1 se věnujeme implementaci algoritmu pro generování klíčů *kryptosystému McEliece*, v 6.3.2 pro šifrování a v 6.3.3 pro dešifrování. Praktickou ukázkou s malými parametry uvádíme v podkapitole 6.3.4 a nakonec v 6.3.5 diskutujeme možná zlepšení bezpečnosti i efektivity implementace.

### 6.3.1 Generování klíčů

Generování klíčů je implementováno dle definice uvedené v kapitole 1.1. Při generování klíčů je třeba vygenerovat lineární samoopravný kód opravující zvolený počet chyb  $t$ , dále náhodnou *regulární*  $k \times k$  matici  $S$  a náhodnou *permutační*  $n \times n$  matici  $P$ . Z generující  $k \times n$  matice  $G$  zvoleného kódu sestojíme veřejný klíč, matici  $\hat{G}$ , která vznikne součinem matic  $S$ ,  $G$  a  $P$ .

---

#### Algoritmus 10 Generování klíčů pro McEliece

---

```

1: function GENERUJMCELTICE[  $m, t$  ]
2:    $n \leftarrow 2^m$ ;  $k \leftarrow n - tm$  ▷ Rozměry matic dle Goppa kódu
3:    $GoppaKod \leftarrow generujGoppaKod[m, t]$ 

4:    $G \leftarrow GoppaKod[[1]]$ 
5:    $S \leftarrow nahodnaRegularniMatice[k]$ 
6:    $P \leftarrow nahodnaPermutacniMatice[n]$ 

7:    $\hat{G} \leftarrow Dot[S, G, P, Modulus \rightarrow 2]$  ▷ Násobení matic
   // Předpočítání inverzí
8:    $invS \leftarrow Inverse[S, Modulus \rightarrow 2]$ 
9:    $invP \leftarrow Inverse[P, Modulus \rightarrow 2]$ 

10:   $soukromyKlic \leftarrow \{GoppaKod, invS, invP\}$ 
11:   $verejnyKlic \leftarrow \{\hat{G}\}$ 
12:   $parametry \leftarrow \{n, k, t\}$ 
13:  return  $\{soukromyKlic, verejnyKlic, parametry\}$ 
14: end function

```

---

Použitým lineárním kódem jsou *ireducibilní binární Goppa kódy*, kterými jsme se zabývali v minulé kapitole. Díky tomu jsou parametry *kryptosystému* (respektive kódu) jednoznačně určené volbou parametrů  $m$  a  $t$ , tedy  $(n, k, t) = (2^m, 2^m - tm, t)$ .

### Generování regulární matice

Generování regulární matice je realizováno velmi primitivním způsobem. Pomocí vnitřní funkce `RandomChoice` se vygeneruje náhodná  $k \times k$  matice a poté se testuje, jestli je *regulární* (funkcí `MatrixRank`). Náhodné matice se tak opakovaně generují dokud nenalezneme matici regulární.

### Generování permutační matice

Permutační matice je vygenerována náhodným permutováním jednotkové matice  $\mathbb{I}_n$  pomocí vnitřní funkce softwaru Mathematica `RandomSample`.

### Inverze a násobení matic

Násobení matic, stejně jako jejich inverze, je v tomto algoritmu realizováno pomocí vnitřních funkcí `Dot` respektive `Inverse`. Volitelný parametr `Modulus` určuje, že výpočet jednotlivých prvků se provádí v daném modulu respektive nad tělesem  $GF(2)$ .

## 6.3.2 Šifrování

Jak bylo zmíněno výše, šifrování probíhá standardním způsobem, který jsme ukázali v kapitole 1.2. Toto šifrování je vlastně prostým násobením matice vektorem a přičtení náhodného chybového vektoru s *Hammíngovou* vahou  $t$ .

---

### Algoritmus 11 Šifrování McEliece

---

```
1: function SIFRUJMCELTCE[  $m, \hat{G}, parametry : \{n, k, t\}$  ]  
    // Chybový vektor délky  $n$   
2:    $z \leftarrow nahodnyChybovyVektor[n, t]$   
    // Zpráva  $m$  délky  $k$   
3:    $b \leftarrow Dot[m, \hat{G}, Modulus \rightarrow 2]$  ▷ Maticové násobení  
    // Výsledný šifrový text délky  $n$   
4:    $c \leftarrow Mod[b + z, Modulus \rightarrow 2]$   
5:   return  $c$   
6: end function
```

---

### Násobení matic a sčítání vektorů

Stejně jako u generování kódu je zde pro násobení matice vektorem použito vnitřní funkce `Dot`. Aplikování chybového vektoru je zde možné realizovat jako prosté sčítání dvou vektorů – listů – modulo 2.



### Náhodný chybový vektor

Náhodný vektor *váhy*  $t$  je sestaven pomocí náhodného permutování vektoru s  $t$  jedničkami a  $n - t$  nulami vnitřní funkcí `RandomSample`.

### 6.3.3 Dešifrování

Dešifrování je opět implementováno tak, jak jsme tento algoritmus popsali v kapitole 1.2. Nejdůležitější částí tohoto algoritmu je dekódování přijatého slova, které je realizováno dekódováním, které jsme popsali v implementaci *binárních Goppa kódů*.

---

#### Algoritmus 12 Dešifrování McEliece

---

```

1: function DESIFRUJMCHELIECE[  $c$ , soukromyKlic, parametry :  $\{n, k, t\}$  ]
2:   {GoppaKod, invS, invP}  $\leftarrow$  soukromyKlic
   // Šifrový text  $c$  má délku  $n$ ; invertování permutace
3:    $\hat{c} \leftarrow \text{Dot}[c, \text{invP}, \text{Modulus} \rightarrow 2]$ 
   // Dekódování zprávy; chybový vektor zde není potřeba
4:    $\hat{m} \leftarrow \text{dekodujBinarniGoppaKod}[\hat{c}, \text{GoppaKod}][[1]]$ 
   // invertování matice  $S$ 
5:    $m \leftarrow \text{Dot}[\hat{m}, \text{invS}, \text{Modulus} \rightarrow 2]$ 
6:   return  $m$ 
7: end function

```

---

### 6.3.4 Ukázka

V této sekci uvádíme krátkou ukázkou vygenerování klíčů s parametry  $m = 3$  a  $t = 2$ . Ukázky s většími parametry i s příkladem použití jsou připravené v souboru `mceliece.nb`.

#### Příklad:

Mějme *ireducibilní binární Goppa kód*  $\Gamma$  s parametry  $(n, k, t) = (8, 2, 2)$  s generující maticí  $G$ :

$$G = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

**Generování klíčů:** Při generování klíčů vygenerujeme náhodnou regulární matici  $S$  a náhodnou permutační matici  $P$ :

$$S = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \quad P = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

*Veřejný klíč* – matici  $\hat{G}$  – sestrojíme vynásobením těchto tří matic:

$$\hat{G} = SGP = \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{pmatrix}$$

*Soukromým klíčem* pak je kód  $\Gamma$ , a předpočítané matice  $S^{-1}$  a  $P^{-1}$ .

$$S^{-1} = \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} \quad P^{-1} = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

**Šifrování:** Mějme otevřený text  $m = (1\ 1)$ . Při šifrování vygenerujeme náhodný chybový vektor  $z$  délky  $n = 8$  a *váhy*  $t = 2$  a zašifrujeme dle předpisu:

$$c = m\hat{G} + z = (1\ 1) \begin{pmatrix} 1 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{pmatrix} + (1\ 1\ 0\ 0\ 0\ 0\ 0\ 0) \\ c = (1\ 0\ 1\ 0\ 0\ 1\ 1\ 1)$$

**Dešifrování:** Pro dešifrování zprávy  $c$  nejdříve provedeme inverzi permutace:

$$\hat{c} = cP^{-1} = (0\ 1\ 1\ 0\ 1\ 1\ 1\ 0)$$

a tento vektor dekódujeme kódem  $\Gamma$ :

$$\hat{m} = D_{\Gamma}(\hat{c}) = (0\ 1)$$

Takto dekódovanou zprávu již jen vynásobíme inverzí  $S$  a získáme tak původní otevřený text  $m$ :

$$m = mS^{-1} = (0\ 1) \begin{pmatrix} 1 & 0 \\ 1 & 1 \end{pmatrix} = (1\ 1)$$

### 6.3.5 Možná zlepšení

Pro zrychlení výpočtů, zvýšení zabezpečení implementace a snížení velikosti klíčů by bylo vhodné se věnovat jistým zlepšením, které popíšeme v následujících odstavcích.

#### Generování ireducibilního polynomu

Ač jsme již tuto skutečnost popsali v kapitole věnující se implementaci *Goppa kódů* 6.2, je vhodné ji kvůli bezpečnosti znovu zdůraznit. V této implementaci používáme předgenerované ireducibilní polynomy, které definují použitý *Goppa* kód. Pro praktické použití je třeba implementovat algoritmus, který bude ireducibilní polynomy generovat náhodně.

### Generování regulární matice

Při generování klíčů je potřeba náhodná regulární matice  $S$ . Tuto matici získáme tak, že generujeme náhodné matice, dokud nenalezneme matici regulární. Jak jsme již popsali v kapitole 1.1, existují algoritmy, které dokáží vygenerovat regulární matici efektivnějším způsobem [35]. Nicméně měření tohoto primitivního způsobu generování matice potvrdilo tvrzení z [20], že v průměru je třeba vygenerovat přibližně 3 náhodné matice, abychom nad  $GF(2)$  našli matici regulární.

### Permutační matice

Matici  $P$  (respektive její inverzi) lze uchovat mnohem kompaktnějším způsobem – jak jsme zmínili hned v 1 kapitole. Nicméně pro demonstrační účely jsme ponechali tuto permutaci ve formě řídké  $n \times n$  matice. V softwaru *Mathematica* bychom permutaci mohli efektivněji uložit ve formě seznamu permutovaných prvků 1 až  $n$  či ve formě tzv. *cyklů*. Vygenerovat takovou formu permutace je možné pomocí funkcí `RandomSample` a `PermutationCycles` a aplikaci této permutace provedeme funkcí `Permute` místo použitého maticového násobení.

### Systematická forma veřejného klíče

Nejjednodušší snížení velikosti *veřejného* klíče je převést matici  $\hat{g}$  do tzv. systematického tvaru. Touto metodou jsme se zabývali v kapitole 5.1. Pokud převedeme matici do tohoto tvaru, tak prvních  $k$  sloupců matice  $\hat{G}$  tvoří jednotkovou matici  $\mathbb{I}_k$  a není třeba tuto část matice vůbec uvádět. Je tedy možné ušetřit  $k^2$  hodnot – bitů. Aby však nebyla snížena bezpečnost *kryptosystému*, je nutné tuto variantu zkombinovat s *CCA2* odolnou konverzí.

### CCA2 odolná konverze

Jak bylo detailně ukázáno v kapitole 4.3, základní varianta použití kryptosystému *McEliece* obsahuje mnoho slabin, které lze zneužít při útoku s voleným šifrovým textem. Pro praktické použití při šifrování je potřeba implementovat šifrování dle *CCA2 odolné* konverze, jakou je například konverze *Kobara-Imai*  $\gamma$ , které jsme se věnovali v kapitole 5.2. Cílem této práce bylo přinést pouze ukázkovou implementaci pro získání představy o *kryptosystému* a proto nebyl kladen důraz na implementaci této konverze.



## Analýza složitosti

V této poslední kapitole se věnujeme analýzou naší implementace kryptosystému *McEliece*, kterou jsme popsali v 6. kapitole. V podkapitole 7.1 provedeme analytické odvození (přibližných) velikostí vygenerovaných klíčů v závislosti na zadaných parametrech  $m$  a  $t$ . V podkapitole 7.2 uvedeme výsledky a závěry z experimentálního měření časových složitostí algoritmů použitých v kryptosystému *McEliece*.

### 7.1 Velikosti klíčů

Velikost klíčů (respektive počet jejich hodnot) je možné velmi přesně určit analyticky. Veřejným klíčem je matice  $\hat{G}$  a soukromým klíčem je kód  $\Gamma$  a inverze matic  $P$  a  $S$ . Veřejnými parametry je pouze trojice čísel  $(n, k, t)$ , které jsou i pro reálné implementace reprezentovány v řádu desítek bitů, a v porovnání s velikostí matic jsou tedy zanedbatelné.

Velikosti budeme vyjadřovat v počtu bitů určujících dané proměnné. Tyto hodnoty jsou ale v naší implementaci objekty typu `Integer`, jelikož v softwaru *Mathematica* nejsme schopni (pohodlně) pracovat s jinými datovými typy. Dále budeme tedy mluvit v jednotkách *hodnot*, čímž budeme rozumět objektu typu `Integer`.

#### 7.1.1 Veřejný klíč

Veřejný klíč je v naší implementaci tvořen  $k \times n$  maticí  $\hat{G}$ . Tato matice má tedy  $nk$  hodnot, kde  $n = 2^m$  a  $k = 2^m - tm$ . V reálných aplikacích algoritmu se hodnota  $k$  pohybuje okolo poloviny hodnoty  $n$  a tedy počet hodnot této matice je přibližně

$$|\hat{G}| = O(2^{2m-1})$$

**Snížení velikosti**

Jak již bylo několikrát zmíněno, tuto matici je možné efektivně uložit v *systematické* formě a velikost klíče tak klesne o  $k$  sloupců této matice na  $(n - k)k$  hodnot. V kapitole 5.1 jsme uvedli, že lze tímto způsobem ušetřit 50-75 % velikosti veřejného klíče.

**7.1.2 Soukromý klíč**

Soukromý klíč je určen *Goppa kódem*  $\Gamma$  a maticí  $S^{-1}$  a  $P^{-1}$ . Matice  $S^{-1}$  je reprezentována  $k^2$  hodnotami a matice  $P^{-1}$   $n^2$  hodnotami.

*Goppa kód*  $\Gamma$  je určen generující maticí  $G$  *Goppovým* polynomem  $g$ , *podporou*  $L$  a předpočítanými *dílčími syndromy*  $S_L$ . Matice  $G$  zabírá stejný prostor jako  $\hat{G}$ . Polynom  $g$  je stupně  $t$  nad konečným tělesem  $GF(2^m)$ , které je určeno ireducibilním polynomem stupně  $m$ . Polynom  $g$  je tedy reprezentován  $m(t + 1)$  hodnotami a vnitřní polynom  $m + 1$  hodnotami. Podpora  $L$  je posloupnost  $n$  prvků z tělesa  $GF(2^m)$  a zabírá tedy  $nm$  hodnot. Syndromy  $S_L$  jsou posloupnost  $n$  prvků z tělesa  $GF(2^m)^t$  a zabírají tedy  $ntm$  hodnot. Celkový součet velikosti *Goppa kódu* je

$$\begin{aligned} |\Gamma| &= O((nk) + (m(t + 1) + (m + 1)) + (nm) + (ntm)) = \\ &= O(nk + (mt + m) + ntm) = \\ &= O(nk + tnm) \end{aligned}$$

Pro běžné parametry platí  $n > k \gg t > m$  a tak nejvýznamnější část určeného Goppa kódu představuje *generující* matice  $G$  a *dílčí syndromy*  $S_L$ .

Velikost soukromého klíče je tak celkově:

$$\begin{aligned} |SK| &= O(|\Gamma| + |S^{-1}| + |P^{-1}|) = \\ &= O(nk + tnm + k^2 + n^2) \end{aligned}$$

**Snížení velikosti**

Velikost uloženého *Goppa kódu* jsme detailně diskutovali v kapitole 6.2.4. V soukromém klíči by navíc nebylo nutné ukládat celou matici  $G$ , která je zde jen kvůli provedení invertování zakódování, které – jak jsme zdůvodnili ve výše zmíněné kapitole – by stačilo provést na základě výběru správných dimenzí. Tento výběr je možné předpočítat a mohli bychom ho v soukromém klíči uložit místo celé matice  $G$ .

Dalším významným ušetřením velikosti klíče je uložení *permutační* matice  $P$  ve formě *permutace* (viz 6.3.5). Touto změnou bychom místo  $n^2$  hodnot matice  $P$  používali  $n$  prvkovou permutaci (neboli  $n$  hodnot).

## 7.2 Experimentální výsledky

V rámci analýzy naší implementace bylo provedeno měření časových složitostí jednotlivých algoritmů pro *generování klíčů*, *šifrování* i *dešifrování*.

Jelikož generování klíčů i dešifrování jsou komplexní algoritmy, skládající se z několika podproblémů, provedli jsme měření i těchto jednotlivých kroků. Měřením těchto podproblémů jsme byli schopni lokalizovat nejnáročnější části výpočtu a diskutovat možná zlepšení.

Aby bylo možné získat přehled o časové náročnosti dílčích výpočtů, bylo nutné lehce upravit zdrojové kódy některých algoritmů. To obsahuje především přidání funkcí `AbsoluteTiming` a navracení hodnot takto spočítaných časů společně s původními návratovými hodnotami.

**Poznámka o umístění souborů:** Z důvodu přehlednosti a zachování původního kódu byly tyto úpravy prováděny ve vlastní *větví mereni* ve verzovací systém *git*. Na příloženém disku jsou soubory z této větve umístěny do podadresáře *mereni*.

### 7.2.1 Způsob měření

Pro měření časových závislostí jsme měřili dobu trvání jednotlivých částí výpočtu pro různé parametry  $m$  a  $t$ . Jak je vidět například na obrázku 7.1, generování klíčů pro  $m = 8$  trvá až 260 sekund a proto bylo měření prováděno pro hodnoty  $m = 3$  až  $m = 8$ . Pro tyto hodnoty  $m$  bylo provedeno měření pro všechny hodnoty  $t$  od  $t = 2$  až po  $t_{max}$ , kde  $t_{max}$  je největší platné  $t$  – tedy pro které platí, že  $k > 0 \Rightarrow 2^m - mt > 0$ . Z tohoto předpisu je vidět, že hodnota  $t_{max}$  roste řádově exponenciálně v závislosti na parametru  $m$ .

Pro každou dvojici parametrů  $m$  a  $t$  bylo opakovaně provedeno *generování klíčů*, *šifrování* i *dešifrování*. Výsledná uložená hodnota naměřeného času dané části algoritmu je určena jako průměr z opakovaně naměřených hodnot.

Logika tohoto prováděného experimentu je zachycena ve zdrojovém kódu *mereni.m*. Výsledná data z prováděného měření se ukládají na disk (do podadresáře *data*). Naměřená data jsou tak k dispozici i po ukončení výpočtu (programu) a je tedy možné tento *balík* spouštět dávkově.

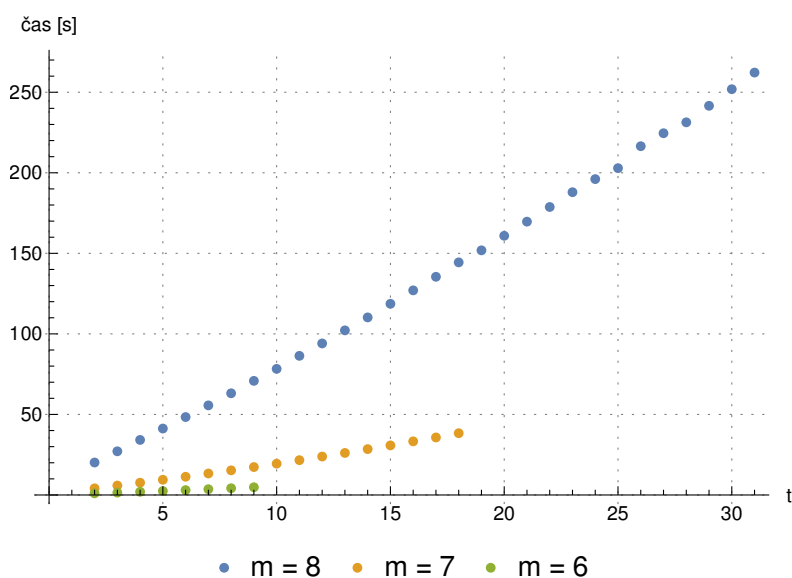
Měření bylo provedeno na stolním počítači s čtyřjádrovým procesorem *Intel i5-6500* na frekvenci *3.2 Ghz* a *16 GB RAM*. Výpočetní kapacita této konfigurace nebyla při měření plně využita, protože implementace využívá pouze jedno vlákno (respektive jádro procesoru) a proces výpočetního jádra *Wolfram Mathematica – WolframKernel* – zabíral i při nejnáročnějších výpočtech řádově 130 MB paměti.

### 7.2.2 Naměřená data

Naměřená data jsou ve formátu pro software *Mathematica* uložena v souboru `data/data.txt`. Z těchto dat jsme vykreslili grafy závislosti času na parametru  $t$  a na parametru  $m$ . Poslední skupina grafů naznačuje procentuální poměr času strávený na dílčích krocích jednotlivých algoritmů.

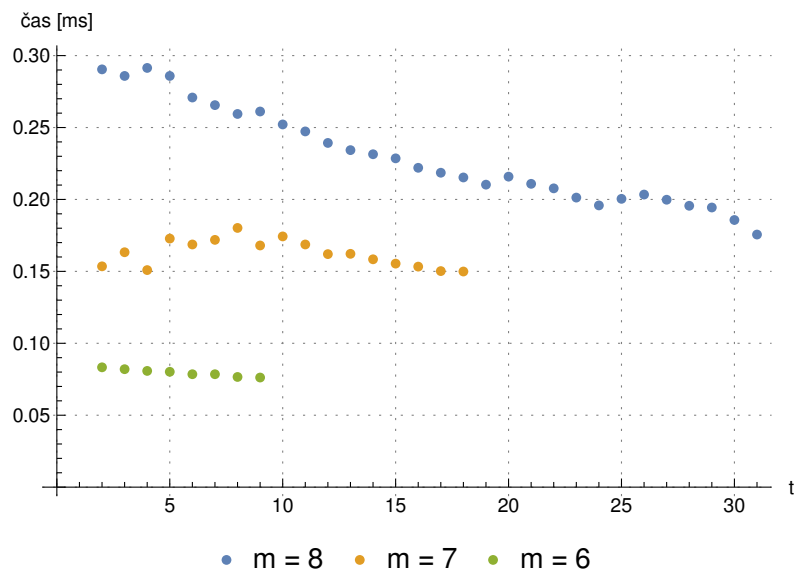
#### Závislost na parametru $t$

První skupina grafů ukazuje časovou závislost na parametru  $t$  pro zadanou hodnotu  $m$  pro generování klíčů (obrázek 7.1), šifrování (obrázek 7.2) a dešifrování (obrázek 7.3). V grafech jsou znázorněny jen hodnoty pro  $m \geq 6$ , protože platných hodnot  $t$  je pro menší  $m$  málo a ani by v grafech nebyly vidět.

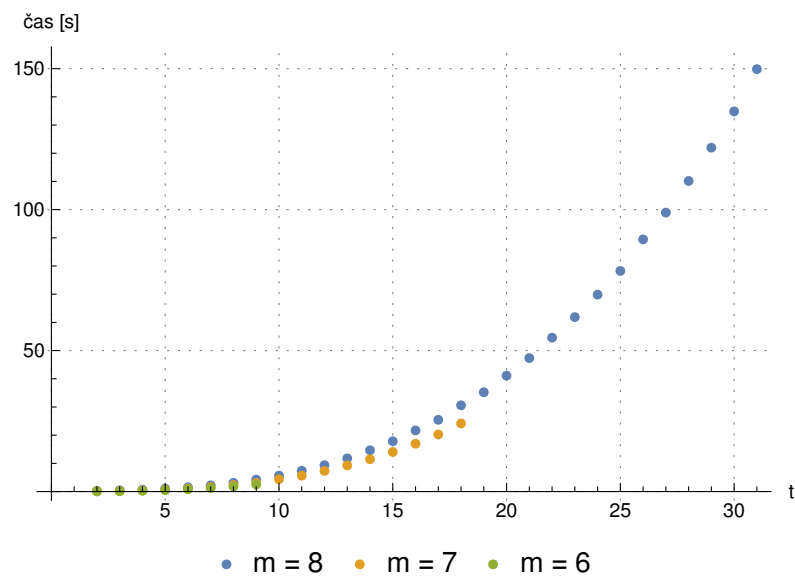


Obrázek 7.1: Závislost doby generování klíčů na parametru  $t$  (při různých volbách  $m$ )





Obrázek 7.2: Závislost doby šifrování zprávy na parametru  $t$  (při různých volbách  $m$ )



Obrázek 7.3: Závislost doby dešifrování zprávy na parametru  $t$  (při různých volbách  $m$ )

**Závislost na parametru  $m$** 

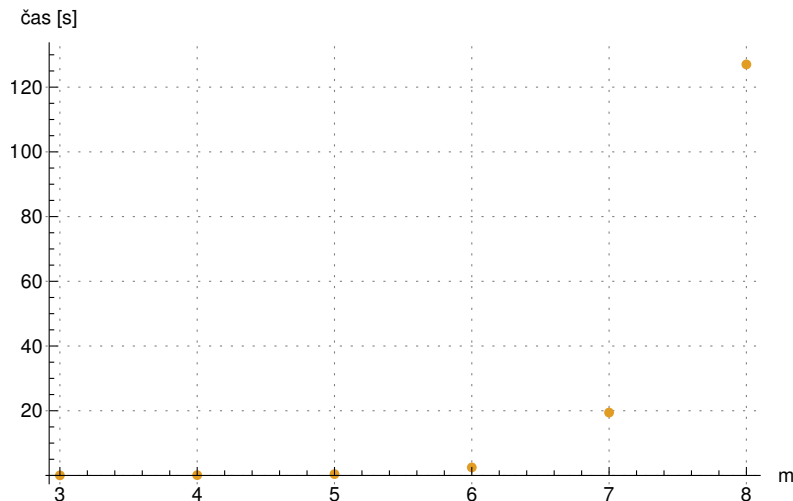
Ve druhé skupině grafů jsme znázornili časovou závislost na parametru  $m$ . Pro každé  $m$  jsou platné stále větší hodnoty  $t$  a v praxi se často volí  $t$  tak, aby hodnota  $k$  byla řádově poloviční vůči  $n$ . Z těchto důvodů jsme v tomto grafu vybrali hodnoty  $t$  odpovídající polovině maximální platné hodnoty  $t_{max}$ .

Na obrázku 7.4 je znázorněna časová závislost pro *generování klíčů*, na obrázku 7.5 pro šifrování a na obrázku 7.6 pro dešifrování.

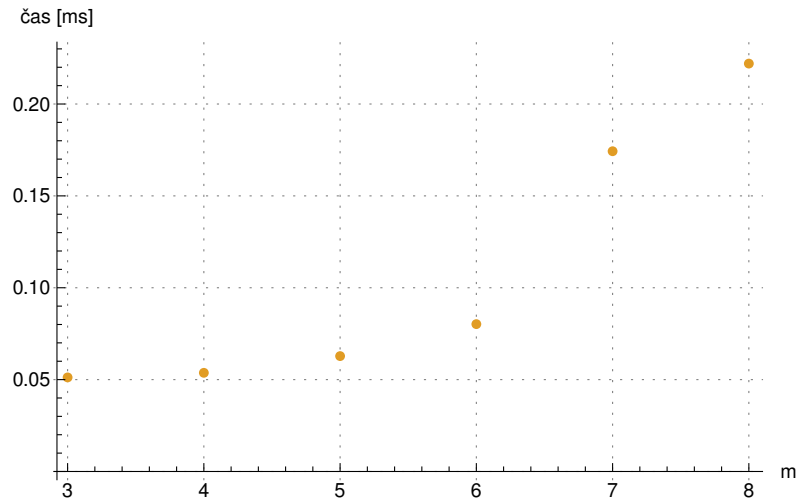
**Poměr významných částí výpočtu**

Jelikož algoritmy pro generování klíčů a dešifrování trvají velmi dlouho, provedli jsme v poslední skupině grafů demonstraci doby trvání dílčích výpočtů těchto komplexních algoritmů. Pro tyto grafy jsme vybrali hodnoty časů pro parametr  $m = 8$  a znázornili závislost na parametru  $t$ .

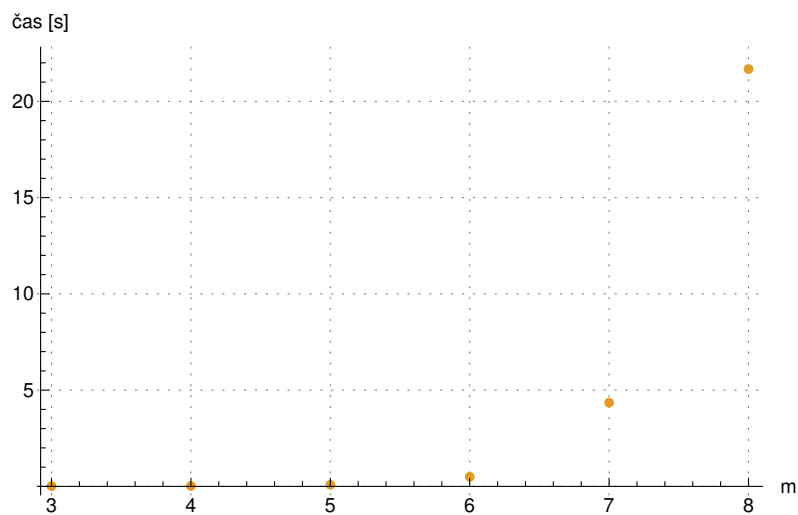
V obrázku 7.7 je vyjádřen poměr částí výpočtu při generování klíčů a to konkrétně diagonální matice  $D$ , spočítání matice  $H$  a předpočítání dílčích syndromů  $S_L$ . V obrázku 7.8 je vyjádřen tento poměr při dešifrování pro vypočítání polynomu  $R$  a pro výsledné hledání kořenů.



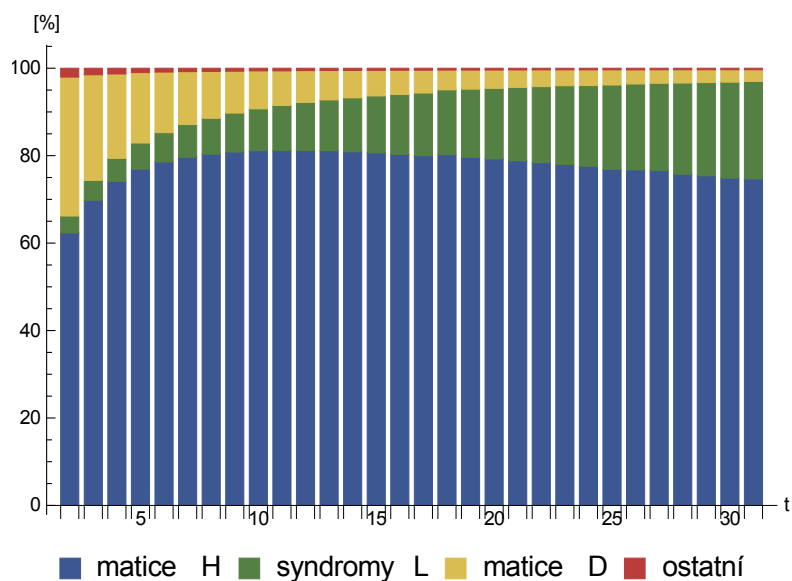
Obrázek 7.4: Závislost doby generování klíčů na parametru  $m$  (parametr  $t$  je zvolen jako polovina maximální možné hodnoty  $t$ )



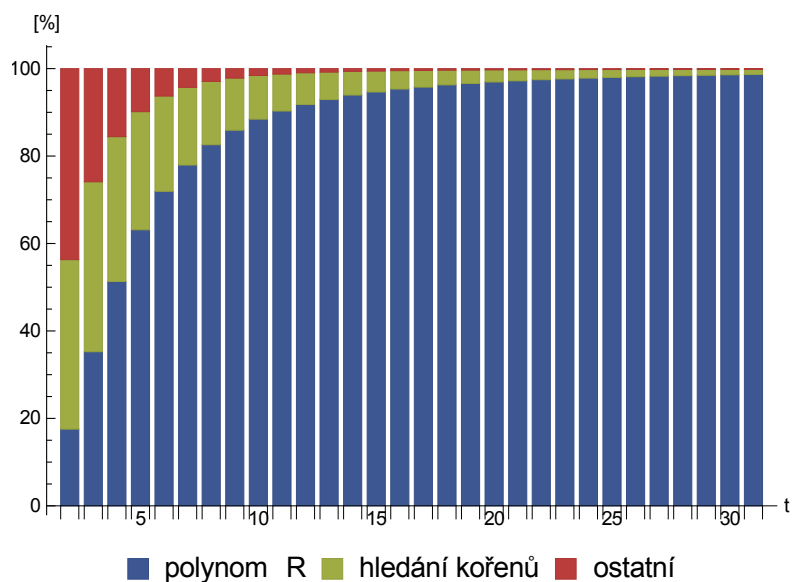
Obrázek 7.5: Závislost doby šifrování zprávy na parametru  $m$  (parametr  $t$  je zvolen jako polovina maximální možné hodnoty  $t$ )



Obrázek 7.6: Závislost doby dešifrování zprávy na parametru  $m$  (parametr  $t$  je zvolen jako polovina maximální možné hodnoty  $t$ )



Obrázek 7.7: Poměr významných částí výpočtu při generování klíčů v závislosti na parametru  $t$  (při  $m = 8$ )



Obrázek 7.8: Poměr významných částí výpočtu při dešifrování zprávy v závislosti na parametru  $t$  (při  $m = 8$ )

### 7.2.3 Závěry z měření

V následujících odstavcích se budeme věnovat diskuzi složitostí dle výsledků měření jednotlivých algoritmů a uvedeným grafům.

#### Závislost na parametru $m$

Závislost doby výpočtu na parametru  $m$  vykazuje u *generování klíčů* (obrázek 7.4) a *dešifrování* (obrázek 7.6) exponenciální charakter. Exponenciální závislost je zapříčiněna tím, že hodnota  $t$  byla volena jako  $t_{max}/2$  a jak jsme zmínili výše,  $t_{max}$  roste s  $m$  *exponenciálně*. Navíc parametr  $m$  určuje velikost vnitřního tělesa  $GF(2^m)$  a *řád* tohoto tělesa (odpovídá počtu prvků *podpory*  $L$ ) roste s  $m$  též exponenciálně. U praktických variant algoritmu tento parametr  $m$  nepřesahuje hodnoty 13 a pohybuje se spíše mezi 11 až 12 (viz kapitola 4.2).

#### Šifrování

U grafů zobrazujících časovou závislost *šifrování* (obrázky 7.2 a 7.5) je nutné zdůraznit, že časová osa grafů pro šifrování je v jednotkách *milisekund*, zatímco u ostatních grafů je v jednotkách *sekund*. Šifrování je u kryptosystému *McElice* řádově rychlejší (to potvrzují i tabulky v kapitole 4.2), jelikož se jedná o prosté násobení matice vektorem – respektive sečtení odpovídajících řádků matice  $\hat{G}$  – a sečtení dvou vektorů. Malé výkyvy v naměřené závislosti mohou být způsobeny tím, že při určitých rozměrech matice se může lépe využívat prostorová lokalita paměťového systému počítače.

Je zřejmé, že doba šifrování klesá v závislosti na parametru  $t$ , protože počet řádků šifrovací matice  $\hat{G}$  klesá s rostoucím  $t$  ( $k = n - tm$ ).

#### Dešifrování

Z grafu závislosti doby dešifrování na parametru  $t$  (obrázek 7.3) je zřejmá *polynomiální* složitost tohoto algoritmu. V obrázku 7.8 je pak zobrazený poměr času výpočtů dílčích kroků dešifrování. Samotné dešifrování – neboli násobení matic  $P^{-1}$  a  $S^{-1}$  – není asymptoticky významné. Nejvíce času trvá dekodování slova  $\hat{c}$  a to konkrétně vypočítání polynomu  $R$  a hledáním kořenů.

Dle [15] má být nejnáročnějším krokem *Pattersonova* dekodovacího algoritmu právě hledání kořenů ale ze znázorněného poměru v obrázku 7.8 vyplývá, že vypočítání odmocniny (polynomu  $R$ ) v našem případě trvá (pro rozumně velká  $t$ ) mnohem déle. Na tento krok algoritmu nebyl kladen důraz v žádné z publikací, ze kterých jsme čerpali a tak je tento poznatek může sloužit jako podnět pro další zkoumání.

V kapitole 6.1.4 jsme se věnovali možnou implementací operací s koeficienty z tělesa  $GF(2^m)$  pomocí *mocnin primitivního prvku* tohoto tělesa. Toto vylepšení může zkrátit výpočet druhé odmocniny polynomu  $R$  z  $GF(2^m)^t$ .

**Generování klíčů**

Při porovnání složitostí prezentovaných grafů na obrázcích v této kapitole je vidět, že algoritmus pro *generování klíčů* je jasně nejpomalejší z prezentovaných algoritmů. V grafu na obrázku 7.1 je patrná polynomiální (dalo by se s dominancí lineární složky) závislost doby generování klíčů na parametru  $t$ . Při detailnějším pohledu na poměr jednotlivých kroků algoritmu v obrázku 7.7 vidíme, že nejdelším dílčím výpočtem je výpočet matice  $H$ . Druhý zajímavý poznatek je rostoucí procentuální zastoupení výpočtu syndromů  $S_L$ . Opět se jedná o dílčí výpočty při generování *Goppa kódu* a ukázalo se tak, že generování matic  $P$  a  $S$  nezastupuje ve výpočtu – při těchto naměřených parametrech  $m$  a  $t$  – významnou část.

Matici  $H$  počítáme prostým násobením dvou matic  $D$  a  $V$ . Matice  $D$  má pro dané  $m$  konstantní velikost  $n \times n \Rightarrow 2^m \times 2^m$  prvků. Počet řádků matice  $V$  je  $t$  a velikost této matice tak roste *lineárně* s parametrem  $t$ . Jelikož jednotlivé prvky matic jsou z tělesa  $GF(2^m)$ , nebylo možné využít vnitřní implementace maticového násobení softwaru *Mathematica* a museli jsme implementovat toto násobení *prostou* metodou pomocí tří *cyklů*. Existují asymptoticky rychlejší algoritmy pro (obecné) násobení matic<sup>23</sup>, nicméně v tomto konkrétním případě se násobí matice  $V$  *diagonální* maticí  $D$  – každý řádek  $V$  je vždy násoben pouze jedním nenulovým prvkem z  $D$ . Za této podmínky je možné násobení implementovat jednodušším způsobem.

Výpočet syndromů  $S_L$  je výpočet *inverzí* v tělese  $GF(2^m)^t$ , kde  $t$  tedy určuje počet koeficientů polynomů v tomto tělese (respektive *řád* tělesa).

Výpočet matice  $H$  nad tělesem  $GF(2^m)$ , stejně jako výpočet inverze *syndromů*  $S_L$ , můžeme opět zrychlit počítáním s prvky vnitřního tělesa  $GF(2^m)$  jako *mocninami primitivního prvku* – viz kapitola 6.1.4.

---

<sup>23</sup> Například *Strassenův* algoritmus.

---

## Závěr

Cílem práce bylo dle zadání získat a přinést co nejvíce ucelený pohled na kryptosystém *McEliece* a problematiku kryptografie založené na lineárních kódech obecně. Struktura práce reflektuje jednotlivé položky zadání.

Prezentovali jsme algoritmy pro asymetrické šifrování *McEliece* a též schéma pro sestrojení digitálního podpisu za pomoci příbuzného kryptosystému *Niederreiter*. Uvedli jsme problematiku zabývající se *ireducibilními binárními Goppa kódy*, ze kterých kryptografie založená na lineárních kódech dodnes vychází.

Provedli jsme hlubokou rešerši publikovaných analýz tohoto *kryptosystému* i existujících útoků. Dále jsme se též věnovali praktickým aspektům v moderních implementacích a metodám na zkrácení klíčů u základní varianty *McEliece* s *binárními Goppa kódy*, tak i v moderních variantách za použití alternativních kódů.

Pro praktické ukázky *kryptosystému McEliece* byla implementována varianta – dle původní definice *Roberta McEliece* – v softwaru *Wolfram Mathematica*. Byly připraveny příklady použití (šifrování) a demonstrace charakteru klíčů, průběhu jejich generování a též průběhu algoritmů pro šifrování a dešifrování.

Na této implementaci bylo provedeno základní měření časových složitostí algoritmů a v této oblasti zbývá jistě mnoho prostoru pro hlubší analýzu. Z výsledků tohoto měření je vidět, na které podproblémy je dobré soustředit další studium a navrhli jsme konkrétní zlepšení této implementace. Velikost klíčů v této implementaci jsme určili analyticky a přinesli jsme návrhy na snížení velikosti dle již známých postupů. Tyto poznatky poslouží při budoucím pokračování ve zlepšování implementace.

V rámci implementace vznikly *balíky* – knihovny – pro výpočty s tzv. *rozšířenými tělesy*, na nich postavené *ireducibilní binární Goppa kódy* a tyto kódy využívající implementace *McEliece*. Každý z těchto tří balíků je možné využít pro další zkoumání kryptosystému *McEliece*, ale také v jiných aplikacích

a rovněž jako podpůrné materiály ve výuce obecné algebry či teorie kódování.

V přílohách jsme připravili základní definice a pojmy z oblasti obecné algebry a teorie kódování. Tyto teoretické poznatky jsou nezbytné pro pochopení problematiky týkající se *kryptografie* založené na lineárních kódech. Pro případná ujasnění některých témat jsou k práci přiloženy.

Přínos této práce je v neposlední řadě v širokém přehledu poznatků i zkušeností z praktické implementace. Práce tak může sloužit jako výchozí bod pro pokračování studování této oblasti asymetrických šifer. Dle našeho názoru tak bylo zadání splněno v plném rozsahu.



---

## Literatura

- [1] Robert J. McELIECE, A Public-Key Cryptosystem Based on Algebraic Coding Theory v *JPL Deep Space Network Progress Report*, strany 114-116. 1978. Dostupné online [http://ipnpr.jpl.nasa.gov/progress\\_report2/42-44/44N.PDF](http://ipnpr.jpl.nasa.gov/progress_report2/42-44/44N.PDF)
- [2] Jiří ADÁMEK. *Kódování*. Edice Matematika pro vysoké školy technické. SNTL, 1989.
- [3] Elwyn R. BERLEKAMP, Robert J. McELIECE, Henk C. A. van TILBORG. On the Inherent Intractibility v *IEEE Transactions of Information Theory*, vol. IT-24, No. 3, strany 384-386. IEEE, květen 1978.
- [4] Elwyn R. BERLEKAMP. Goppa Codes v *IEEE Transactions on Information Theory*, vol. 19, strany 590-592. IEEE, 1973. Dostupné online <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1055088>
- [5] Elwyn R. BERLEKAMP. Factoring polynomials over large finite fields v *Mathematics of Computation*, strany 713-755. 1970.
- [6] Daniel J. BERNSTEIN, Tanja LANGE, Christiane PETERS. Attacking and Defending the McEliece Cryptosystem v *Post-Quantum Cryptography*, strany 31-46. Springer Berlin Heidelberg 2008. Dostupné online [http://link.springer.com/chapter/10.1007/978-3-540-88403-3\\_3](http://link.springer.com/chapter/10.1007/978-3-540-88403-3_3)
- [7] Daniel J. BERNSTEIN. List decoding for binary Goppa codes v *Coding and Cryptology*, vol. 6639, strany 62-80. Springer Berlin Heidelberg 2011. Dostupné online [http://link.springer.com/chapter/10.1007/978-3-642-20901-7\\_4](http://link.springer.com/chapter/10.1007/978-3-642-20901-7_4)
- [8] Daniel J. BERNSTEIN, Johannes BUCHMANN, Erik DAHMEN. *Post-Quantum Cryptography*. ISBN 978-3-540-88701-0. Springer Berlin Heidelberg, 2009.

- [9] T. A. BERSON. Failure of the McEliece public-key cryptosystem under message-resend and related-message attack v *Advances in Cryptology-CRYPTO '97*, vol. 1294, strany 213-200, Springer Berlin, 1997.
- [10] E. F. BRICKELL, A. M. ODLYZKO. Cryptanalysis: a survey of recent results v *Proceedings of the IEEE*, vol. 76, strany 578-593. IEEE, 1988. Dostupné online <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=4443>
- [11] Anne CANTEAUT, Florent CHABAUD. Improvements of the Attacks on Cryptosystems Based on Error-Correcting Codes, v *Research Report LIENS-95-21*. École Normale Supérieure, 1995 Dostupné online <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.32.1645>
- [12] Robert T. CHIEN. Cyclic decoding procedures for Bose- Chaudhuri-Hocquenghem codes v *IEEE Transactions on Information Theory*, vol. 10, strany 357-363. IEEE, 1964. Dostupné online <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1053699>
- [13] Nicolas T. COURTOIS, Matthieu FINIASZ, Nicolas SENDRIER. How to Achieve a McEliece-Based Digital Signature Scheme v *Advances in Cryptology - ASIACRYPT 2001*, strany 157-174. Springer Berlin Heidelberg, 2001. Dostupné online [http://link.springer.com/chapter/10.1007%2F3-540-45682-1\\_10](http://link.springer.com/chapter/10.1007%2F3-540-45682-1_10)
- [14] Hang DINH, Cristopher MOORE, Alexander RUSSELL. McEliece and Niederreiter Cryptosystems That Resist Quantum Fourier Sampling Attacks v *Advances in Cryptology - CRYPTO 2011*, vol. 6841, strany 761-779. Springer Berlin Heidelberg, 2011. Dostupné online [http://link.springer.com/chapter/10.1007%2F978-3-642-22792-9\\_43](http://link.springer.com/chapter/10.1007%2F978-3-642-22792-9_43)
- [15] Daniela ENGELBERT, Raphael OVERBECK, Arthur SCHMIDT. A Summary of McEliece-Type Cryptosystems and their Security v *Journal of Mathematical Cryptology*. IACR 2006. Dostupné online <http://eprint.iacr.org/2006/162>
- [16] Jean-Charles FAUGÈRE, Ayoub OTMANI, Ludovic PERRET, Jean-Pierre TILICH. Algebraic Cryptanalysis of McEliece Variants with Compact Keys v *Advances in Cryptology - EUROCRYPT 2010*. Springer Berlin Heidelberg, 2010. Dostupné online [http://link.springer.com/chapter/10.1007%2F978-3-642-13190-5\\_14](http://link.springer.com/chapter/10.1007%2F978-3-642-13190-5_14)
- [17] Jean-Charles FAUGRE, Ayoub OTMANI , Ludovic PERRET, Frederic de PORTZAMPARC, Jean-Pierre TILICH. *Structural Cryptanalysis of McEliece Schemes with Compact Keys*. IACR Cryptology ePrint Archive, 2014. Dostupné online <https://eprint.iacr.org/2014/210.pdf>

- 
- [18] Shuhong GAO, Daniel PANARIO. Tests and Constructions of Irreducible Polynomials over Finite Fields v *Foundations of Computational Mathematics*, strany 346-361. Springer Berlin Heidelberg, 1997. Dostupné online [http://link.springer.com/chapter/10.1007%2F978-3-642-60539-0\\_27](http://link.springer.com/chapter/10.1007%2F978-3-642-60539-0_27)
- [19] Valery D. GOPPA. A New Class of Linear Correcting Codes v *Problemy Peredachi Informatsii*, vol. 6, strany 24-30. 1970.
- [20] Stefan HEYSE. *Code-based Cryptography: Implementing the McEliece Scheme on Reconfigurable Hardware*. Diplomová práce. Ruhr-University Bochum, 2009.
- [21] A. Al JABRI. A Statistical Decoding Algorithm for General Linear Block Codes v *Cryptography and Coding*, vol. 2260, strany 1-8. Springer Berlin Heidelberg, 2001. Dostupné online [http://link.springer.com/chapter/10.1007%2F3-540-45325-3\\_1](http://link.springer.com/chapter/10.1007%2F3-540-45325-3_1)
- [22] Kazukuni KOBARA, Hideki IMAI. Semantically Secure McEliece Public-Key Cryptosystems – Conversions for McEliece PKC v *Public Key Cryptography*, vol. 1992, strany 19-35. Springer Berlin Heidelberg, 2001. Dostupné online <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.5.9666>
- [23] Jaroslav KOTIL. *Goppa kódy a jejich aplikace*. Diplomová práce. Matematicko-fyzikální fakulta Univerzity Karlovy, Praha, 2013.
- [24] Miroslav KRATOCHVÍL. *Implementation of cryptosystem based on error-correcting codes*. Bakalářská práce. Matematicko-fyzikální fakulta Univerzity Karlovy, Praha, 2013.
- [25] P. J. LEE, E. F. BRICKELL. An Observation on the Security of McEliece's Public-Key Cryptosystem v *Advances in Cryptology – EUROCRYPT '88*, strany 275-280. Springer Berlin Heidelberg, 1988. Dostupné online [http://link.springer.com/chapter/10.1007%2F3-540-45961-8\\_25](http://link.springer.com/chapter/10.1007%2F3-540-45961-8_25)
- [26] J. S. LEON. A probabilistic algorithm for computing minimum weights of large error-correcting codes v *IEEE Transactions on Information Theory*, vol. 34, strany 1354-1359. IEEE, 1988. Dostupné online <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=21270>
- [27] Robert McELIECE. *The Theory of Information and Coding*. Encyclopedia of Mathematics and its Applications, vol. 3. Addison-Wesley, 1977.
- [28] J. G. MERCHAN, S. KUMAR, C. PAAR, J. PELZL. Efficient Software Implementation of Finite Fields with Applications to Cryptography v *Acta Applicandae Mathematicae: An International Sur-*

- vey Journal on Applying Mathematics and Mathematical Applications, Volume 93, strany 3-32. Ruhr-Universitat Bochum, 2006. Dostupné online: <http://www.emsec.rub.de/research/publications/efficient-software-implementation-finite-fields-ap/>
- [29] Rafael MISOCZKI, Paulo S. L. M. BARRETO. Compact McEliece Keys from Goppa Codes v *Selected Areas in Cryptography: 16th Annual International Workshop*, strany 376-392. Springer Berlin Heidelberg, 2009. Dostupné online [http://link.springer.com/chapter/10.1007%2F978-3-642-05445-7\\_24](http://link.springer.com/chapter/10.1007%2F978-3-642-05445-7_24)
- [30] Rafael MISOCZKI, Jean-Pierre TILICH, Nicolas SENDRIER, Paulo S. L. M. BARRETO. MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes v *Information Theory Proceedings*, strany 2069-2073. IEEE, 2013 Dostupné online <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6620590>
- [31] Harald NIEDERREITER. Knapsack-type cryptosystems and algebraic coding theory v *Problems of Control and Information Theory 15*, strany 19-34. 1986
- [32] Christof PAAR, Jan PELZL. *Understanding Cryptography: A Textbook for Students and Practitioners*. Springer-Verlag Berlin Heidelberg, 2010. Dostupné online: <https://www.springer.com/us/book/9783642041006>
- [33] Olga PAUSTJAN. *Post Quantum Cryptography on Embedded Devices: An Efficient Implementation of the McEliece Public Key Scheme based on Quasi-Dyadic Goppa Codes*. Diplomová práce. Ruhr-University Bochum, 2010.
- [34] Nicholas J. PATTERSON, The algebraic decoding of Goppa codes v *IEEE Transactions on Information Theory*, vol. 21, strany 203-207. IEEE 1975. Dostupné online <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1055350>
- [35] Dana RANDALL. *Efficient Generation of Random Nonsingular Matrices*. EECS Department, University of California, 1991. Dostupné online <http://www.eecs.berkeley.edu/Pubs/TechRpts/1991/CSD-91-658.pdf>
- [36] Marek REPKA, Pavol ZAJAC. Overview of the McEliece Cryptosystem and its Security v *Tatra Mountains Mathematical Publications*, vol. 60, strany 57-83. Slovak Academy of Sciences, 2014. Dostupné online <http://www.degruyter.com/view/j/tmmp.2014.60.issue-1/tmmp-2014-0025/tmmp-2014-0025.xml>

- 
- [37] Nicolas SENDRIER. Finding the Permutation Between Equivalent Linear Codes: The Support Splitting Algorithm v *Transactions on Information Theory*, vol. 46. IEEE 2000. Dostupné online <http://ieeexplore.ieee.org/xpl/abstractAuthors.jsp?arnumber=850662>
- [38] J. M. Schanck, W. Whyte, Z. Zhang. Criteria for selection of public-key cryptographic algorithms for quantum-safe hybrid cryptography (Internet-draft). IETF, 2016. Dostupné online <https://datatracker.ietf.org/doc/draft-whyte-select-pkc-qsh/>
- [39] V. M. SIDELNIKOV, S. O. SHESTAKOV. On insecurity of cryptosystems based on generalized Reed-Solomon codes v *Discrete Mathematics and Applications* vol. 2, strany 439-444. Walter de Gruyter 1992. Dostupné online [https://www.researchgate.net/publication/250969195\\_On\\_insecurity\\_of\\_cryptosystems\\_based\\_on\\_generalized\\_Reed-Solomon\\_codes](https://www.researchgate.net/publication/250969195_On_insecurity_of_cryptosystems_based_on_generalized_Reed-Solomon_codes)
- [40] Jacques STERN. A method for finding code words of small weight, v *Coding Theory and Applications*, 3rd International Colloquium, strany 106-113. Springer Berlin Heidelberg, 1988. Dostupné online <http://link.springer.com/chapter/10.1007/BFb0019850>
- [41] Toshiya ITOH, Shigeo TSUJII. A fast algorithm for computing multiplicative inverses in  $GF(2^m)$  using normal bases v *Information and Computation*, vol. 78, strany 171-177. Academic Press, 1988. Dostupné online <http://www.sciencedirect.com/science/article/pii/0890540188900247>
- [42] Valérie Gauthier UMAÑA, Gregor LEANDER. *Practical Key Recovery Attacks on two McEliece Variants*. IACR Cryptology ePrint Archive, 2009. Dostupné online <https://eprint.iacr.org/2009/509.pdf>
- [43] Yuan XING LI, Robert H. DENG, Xin MEI WANG. On the equivalence of McEliece's and Niederreiter's public-key cryptosystems v *IEEE Transactions on Information Theory*, vol. 40, strany 271-273. IEEE, leden 1994. Dostupné online <http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=272496>
- [44] Alois PLUHÁČEK. *Aritmetika a kódy* (přednášky). České vysoké učení technické v Praze, Fakulta informačních technologií, 2014.
- [45] Martin NOVOTNÝ, Róbert LÓRENCZ, Jiří BUČEK. *Bezpečnost a technické prostředky* (přednášky). České vysoké učení technické v Praze, Fakulta informačních technologií, 2013.
- [46] Róbert LÓRENCZ, Josef KOKEŠ. *Pokročilá kryptologie* (přednášky). České vysoké učení technické v Praze, Fakulta informačních technologií, 2013.

- [47] Jan MAREŠ. *Algebra – Úvod do obecné algebry*. Skripta. ČVUT, 1999.
- [48] Jiří PYTLÍČEK. *Lineární algebra a geometrie*. Skripta. ČVUT, 2008.
- [49] FREE SOFTWARE FOUNDATION, INC.. *GNU bash 4.3* (software). 2014. Dostupné online <http://www.gnu.org/software/bash/>
- [50] Linus TORVALDS, Junio C. HAMANO. *git 2.1* (software). 2016. Dostupné online <https://git-scm.com/>
- [51] THE TEX USERS GROUP. *T<sub>E</sub>X Live 2014* (software). Dostupné online <https://www.tug.org/texlive/>
- [52] WOLFRAM RESEARCH, INC.. *Wolfram Mathematica 10* (software). 2015. Dostupné online <https://www.wolfram.com/mathematica/>
- [53] William STEIN. *SageMath* (software). 2016. Dostupné online <http://www.sagemath.org/>

# Obecná algebra

V této příloze uvedeme pojmy a algoritmy nutné pro práci s *konečnými tělesy* a *polynomy* nad konečnými tělesy (*rozšířená tělesa*). Při popisu je předpokládána znalost základních pojmů z oblasti *algebry*, zejména *lineární*. Definice byly převzaty z [32, 47, 48] a tuto literaturu zároveň doporučujeme pro hlubší studium této problematiky.

**Poznámka:** Algoritmy zmíněné v následujících kapitolách jsou detailně – včetně pseudokódu – popsány v kapitole 6, která se zabývá konkrétní implementací algoritmů a operací.

## A.1 Základní termíny

Pro ujasnění je uvedena definice tělesa:

**Definice 5 (Těleso)** *Nechť  $M$  je neprázdná množina a  $+$  a  $\cdot$  binární operace<sup>24</sup>. Struktura  $T = (M, +, \cdot)$  se nazývá těleso, pokud platí*

1.  $(M, +)$  je komutativní grupa (nazývána aditivní)
2.  $(M \setminus \{0\}, \cdot)$ <sup>25</sup> je grupa (nazývána multiplikativní)
3. Platí (levý i pravý) distributivní zákon:

$$\forall a, b, c \in M : (a(b + c) = ab + ac) \wedge ((b + c)a = ba + ca)$$

*Těleso, které má konečný počet prvků, se nazývá konečné těleso.*

**Věta 1** *Nechť  $T$  je konečné těleso, pak jeho počet prvků (řád) je  $p^n$ , kde  $p$  je prvočíslo a  $n \in \mathbb{N} \wedge n \geq 1$ .*

<sup>24</sup> Pro zjednodušení zápisu je  $\cdot$  často vynecháváno.

<sup>25</sup> Prvek  $0$  je nulový (neutrální) prvek aditivní grupy.

Číslo  $p$  se nazývá *charakteristika*. Navíc platí, že *všechna konečná tělesa* se stejným počtem prvků jsou navzájem *izomorfní*. *Konečné těleso* řádu  $p^n$  je tedy dále označováno jako  $GF(p^n)$  (z anglického *Galois field*, dle francouzského matematika *Évariste Galois*).

## A.2 Reprezentace prvků

Jak ukážeme dále, je vhodné prvky konečného tělesa  $GF(p^n)$  reprezentovat jako *polynomy* s koeficienty z množiny  $\mathbb{Z}_p = \{0, 1, \dots, p-1\}$ , tedy prvek  $a \in GF(p^n)$  lze zapsat:

$$a(x) = \sum_{i=0}^{n-1} a_i x^i, a_i \in \mathbb{Z}_p$$

O takovém polynomu říkáme, že je to *polynom nad tělesem*  $GF(p)$  (*stupně maximálně*  $n-1$ ). Na prvek  $a$  je možné dívat se též jako na vektor či  $n$ -tici koeficientů  $a_i$ :

$$a(x) \cong a \cong (a_{n-1} a_{n-2} \dots a_0) \cong a_{n-1} a_{n-2} \dots a_0$$

Mezi těmito reprezentacemi budeme nadále volně přecházet, jak bude v daném kontextu potřeba<sup>26</sup>.

## A.3 Operace v tělese $GF(p^n)$

V následujících sekcích uvedeme operace potřebné pro počítání s tělesy  $GF(p^n)$ . Konkrétní zvolené algoritmy a jejich implementace je detailně popsána v kapitole 6.

**Poznámka:** Kvůli zobecnění definic budeme v této podkapitole používat označení tělesa  $GF(p)$  jako těleso  $\mathbb{F}$ .

### A.3.1 Sčítání

Sčítání v tělese  $GF(p^n)$  je definováno stejně jako sčítání polynomů, s tím, že sčítání jednotlivých koeficientů je prováděno *modulo*  $p$  (v tělese  $GF(p)$ ):

$$a(x) + b(x) = \sum a_i x^i + \sum b_i x^i = \sum |a_i + b_i|_p x^i$$

---

<sup>26</sup> V některých materiálech se používá i obráceného zápisu  $(a_0 a_1 \dots a_{n-1})$ .



### A.3.2 Násobení

Násobení v tělese  $GF(p^n)$  nelze provádět „po složkách“, jako je tomu u sčítání. U takto definované operace by většina prvků neměla (multiplikativní) *inverzi* a nejednalo by se tak o *těleso*.

Při násobení prvků opět využijeme jejich reprezentace pomocí polynomů. Výsledkem násobení pak je:

$$a(x) \cdot b(x) = \sum_{i=0}^{n-1} a_i x^i \cdot \sum_{i=0}^{n-1} b_i x^i = \sum_{i=0}^{2n-2} \left| \sum_{j+k=i} a_j \cdot b_k \right|_p x^i$$

Jak je naznačeno, násobení i sčítání koeficientů se provádí *modulo*  $p$  (respektive v tělese  $\mathbb{F}$ ).

Kvůli uzavřenosti násobení v tělese je nutné zavést operaci „zbytek po dělení polynomu“  $a(x)$  polynomem  $p(x)$ , neboli  $a(x) \bmod p(x)$ . Navíc pro jednoznačné určení tělesa  $GF(p^n)$  je třeba určit příslušný *ireducibilní* polynom  $p(x)$ , který bude použit při operaci násobení modulo  $p(x)$ .

**Definice 6** Polynom  $p(x)$  nad tělesem  $GF(p)$  je *ireducibilní právě tehdy, když pro každé dva polynomy  $a(x)$  a  $b(x)$  nad  $GF(p)$  platí:*

$$a(x) \cdot b(x) = p(x) \Rightarrow (\deg(a(x)) = 0) \vee (\deg(b(x)) = 0)$$

Jinými slovy pro *ireducibilní* polynom platí, že tento polynom nelze rozložit na polynomy nad  $\mathbb{F}$  stupně větší než 1.

**Příklad** Polynom  $x^3 + x + 1$  je nad tělesem  $GF(2)$  *ireducibilní*, protože neexistuje jeho rozklad na polynomy stupně alespoň 1. Polynom  $x^2 + 1$  není nad tělesem  $GF(2)$  *ireducibilní*, protože:

$$(x + 1) \cdot (x + 1) = x^2 + |1 + 1|_2 x + 1 = x^2 + 1$$

Nyní je možné zavést operaci násobení dvou prvků tělesa jako násobení dvou polynomů *modulo* *zadaný ireducibilní* polynom:

$$a(x) \cdot b(x) = \sum a_i x^i \cdot \sum b_i x^i = \sum \left| \sum_{j+k=i} a_j \cdot b_k \right|_p x^i \bmod p(x)$$

**Poznámka:** Pokud by zvolený  $p(x)$  nebyl *ireducibilní*, jednalo by se o *okruh*, nikoliv o *těleso*, protože by neexistovala *multiplikativní inverze* pro některé prvky a navíc by i existovaly tzv. *dělitelé nuly*.

### A.3.3 Umocňování

Pro rozšíření operací o opakované násobení je vhodné zavést operaci umocňování.

**Definice 7** *Nechť  $a$  je prvkem (libovolného) tělesa  $T$  a číslo  $n \in \mathbb{N}$ . Operace umocňování definujeme následovně:*

$$\begin{aligned} a^0 &= 1 \\ a^n &= \underbrace{a \cdot a \cdot \dots \cdot a}_{n\text{-krát}} \\ a^{-n} &= \left(a^{-1}\right)^n \end{aligned}$$

Pro efektivní výpočet mocniny prvku je vhodné použít algoritmus *Square-and-Multiply*, kde se dílčí operace „square“ a „multiply“ provádí operací  $\cdot$  v daném tělese  $F$ .

### A.3.4 Inverze

Inverzi v grupě lze obecně definovat následovně:

**Definice 8 (Inverze)** *Nechť  $G = (M, \circ)$  je grupa,  $a$  jejím prvkem a  $\mathbb{O}$  jejím neutrálním prvkem. Prvek  $\bar{a}$  je inverzí prvku  $a$ , pokud platí následující rovnice:*

$$a \circ \bar{a} = \mathbb{O}$$

#### Aditivní inverze

Inverze v *aditivní grupě* značíme znaménkem minus „ $-$ “ a je z definice velmi triviální:

$$|a(x) + (-a(x))|_p = 0 \Rightarrow -a(x) = \sum | -a_i |_p x^i$$

Neboli je to aditivní inverze jednotlivých koeficientů *modulo*  $p$  (v tělese  $F$ ).

#### Multiplikativní inverze

Inverze v *multiplikativní grupě* značíme záporným exponentem „ $^{-1}$ “ či symbolem dělení.

$$\left| a(x) \cdot a(x)^{-1} \right|_{p(x)} = \left| \frac{a(x)}{a(x)} \right|_{p(x)} = \mathbf{1}$$

Tuto *multiplikativní inverzi* je třeba počítat *rozšířeným Euklidovým algoritmem pro polynomy (EEA)*, či případně jinými algoritmy, jako je například *algoritmus Itoh-Teechai-Tsujii (ITT)* [45, 41].

*Rozšířený Euklidův algoritmus pro polynomy*, stejně jako v modulární aritmetice (neboli pro tělesa  $GF(p)$ ), stojí na nalezení *Bézoutovy rovnosti*. Pro

výpočet *EEA* je třeba výpočtu dělení polynomů se zbytkem<sup>27</sup>, nicméně v binárních tělesech lze toto dělení nahradit prostým posouváním a „odečítáním“ (respektive sčítáním) prvků.

## A.4 Rozšířená tělesa

V algebře se dá rozšíření těles definovat velmi obecně. Pro účely naší práce nás ale budou ve své podstatě zajímat pouze tělesa  $GF(2^n)^m$ .

**Definice 9** *Rozšíření tělesa:* Necht  $T$  je těleso a  $P$  podmnožina množiny těles  $T$ . Pokud  $P$  tvoří (s původními operacemi) opět těleso, říkáme, že  $P$  je podtělesem  $T$  a zároveň  $T$  je rozšířením tělesa  $P$ .

### Polynomy jako rozšířená tělesa

Na konečná tělesa  $GF(p^n)$  realizované polynomy, jak byly představeny v minulé kapitole je možné se dívat jako na *rozšíření* tělesa  $GF(2)$ . Stejně tak, jako jsme sestrojily polynomy nad  $GF(2^n)$  („polynomy nad polynomy“).

**Definice 10** *Okruh polynomů:* Necht  $\mathbb{F}$  je těleso. Množinu okruhu polynomů  $R = \mathbb{F}[x]$  definujeme jako všechny polynomy s koeficienty z tělesa  $\mathbb{F}$  a operace tohoto okruhu jako klasické operace s polynomy s tím, že operace s koeficienty jsou prováděny v tělese  $\mathbb{F}$ .

Tato definice jistě dává smysl, protože operace v tělese  $\mathbb{F}$  jsou uzavřené a výsledkem sčítání respektive násobení dvou polynomů z  $\mathbb{F}[x]$  vznikne polynom, který opět patří do tohoto okruhu.

V případě zavedení operace *modulo polynom* můžeme zavést násobení stejným způsobem, jako bylo uvedeno v předešlé kapitole.

**Tvrzení 2** *Necht  $\mathbb{F}$  je konečné těleso a  $g \in \mathbb{F}[x]$  ireducibilní polynom stupně  $n$ . Potom množina všech polynomů z  $\mathbb{F}[x]$  stupně menší než  $t$  tvoří s klasickou operací s polynomy  $+$  a s násobením modulo  $g$  konečné těleso  $\mathbb{F}^n$ .*<sup>28</sup>

**Poznámka:** Pokud rozšířením konečného tělesa vznikne opět konečné těleso (dle tvrzení výše), je možné toto těleso opět rozšířit a induktivním krokem tak rozšiřovat tělesa do libovolné „hloubky“. Jinými slovy polynomy s operací modulo (ireducibilní) polynom tvoří opět těleso a jdou tak využít jako koeficienty dalších polynomů.

<sup>27</sup> Někdy uváděno jako dlouhé dělení.

<sup>28</sup> Toto těleso se dá též značit jako (faktorokruh)  $\mathbb{F}[x]/(g)$ , kde  $(g)$  je *ideál* generovaný polynomem  $g$ .

**Použité značení**

Příklady počítání operací v těchto *rozšířených* tělesech jsou uvedeny v kapitole implementace 6.1. Pro ujasnění způsobu značení prvků z těchto těles zde uvedeme příklad tohoto značení.

Máme těleso  $\mathbb{F} = GF(2^3)$  s ireducibilním polynomem

$$g(y) = y^3 + y + 1$$

Tento polynom budeme zkráceně zapisovat jako 1011. Potom polynom  $a$  z okruhu  $\mathbb{F}[x]$  zapisujeme těmito ekvivalentními způsoby:

$$\begin{aligned} a(x) &= (y^2 + 1)x^3 + (y)x + (y + 1) \cong \\ &\cong (101)x^3 + (010)x + (011) \quad \cong \\ &\cong (101)(000)(010)(011) \end{aligned}$$

## Teorie kódování

V této příloze definujeme a vysvětlíme pojmy z teorie kódování, které jsou použité v kryptosystému *McEliece* (kapitola 1). Definice byly čerpané z [44, 2] a pro další studium této problematiky je též doporučeno [27]. V kapitolách B.1 a B.2 uvádíme základní značení a termíny z oblasti samoopravných respektive lineárních kódů.

### B.1 Samoopravné kódy

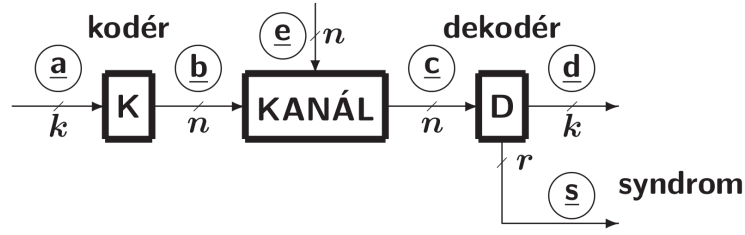
Teorie kódování spadá do oblasti teorie informace a zabývá se způsoby *zakódování* či *reprezentací* zpráv a jejich přenosem.

V této kapitole budeme používat následujícího značení (odpovídá značení na obrázku B.1):

$+$	binární bitová operace <i>XOR</i>
$-$	inverzní operace k $+$ , tedy též <i>XOR</i>
$a$	zpráva délky $k$
$K_{\mathcal{K}}(\cdot)$	operace <i>zakódování</i> kódem $\mathcal{K}$
$b$	zakódovaná zpráva délky $n$ ; $b = K(a)$ a zpravidla platí $n > k$
$e$	chybový vektor délky $n$ vzniklý při přenosu $b$
$c$	přijatá zpráva ( $c = b + e$ )
$D_{\mathcal{K}}(\cdot)$	operace <i>dekódování</i> kódem $\mathcal{K}$
$d$	<i>dekódovaná</i> zpráva; $d = D(c)$
$s$	<i>syndrom</i> přijaté zprávy/chyby (viz dále) délky $r$ ; zpravidla platí $r = n - k$

**Definice 11** (*Binární kód*): *Nechť existuje (prosté) zobrazení  $\mathcal{K}$  z množiny všech možných zpráv a délky  $k$  do množiny kódových slov  $b$  délky  $n$  ( $GF(2)^k \rightarrow GF(2)^n$ ). Pak toto zobrazení nazveme kódem  $\mathcal{K}$  s parametry  $(n, k)$ .*<sup>29</sup>

<sup>29</sup> Obecně lze *kód* definovat jako zobrazení  $\mathcal{L}^k \rightarrow \mathcal{M}^n$ , kde  $\mathcal{L}$  je *abeceda* zpráv délky  $k$  a  $\mathcal{M}$  *abeceda* kódových slov délky  $n$ .



Obrázek B.1: Použité značení v teorii kódování [44]

**Poznámka:** Dále předpokládáme použití pouze *blokových kódů* (dle definice). Dají se definovat i kódy s proměnlivou délkou kódových slov.

Z definice prostého zobrazení vyplývá, že existuje *kódové slovo* pro všechny zprávy a že existuje *inverzní* zobrazení  $\mathcal{K}^{-1}$ . Množina všech kódových slov je jednoznačně určena zobrazením množiny zpráv  $\mathcal{B} = \mathcal{K}(GF(2)^k) = \mathcal{K}(\mathcal{A})$ . Vektory délky  $n$ , které nepatří do množiny  $\mathcal{B}$  nazveme jako *nekódová slova* (vektory). Operaci *zakódování* budeme rozumět aplikaci zobrazení  $\mathcal{K}$  a operaci *dekódování* aplikaci  $\mathcal{K}^{-1}$ , tedy získání původní zprávy z (*kódového*) slova.

**Definice 12** *Hammingova vzdálenost:* Hammingova vzdálenost dvou vektorů  $u$  a  $v$  –  $vzd(u, v)$  či  $H(u, v)$  – je počet rozdílných bitů ve vektorech  $u$  a  $v$ :  $vzd(u, v) = \sum |u_i - v_i|$

*Hammingovu váhu* vektoru  $v$  pak definujeme jako *Hammingovu vzdálenost* vektoru  $v$  od *nulového vektoru* patřícíné délky. Jinými slovy je to počet *nenulových* bitů („jedniček“) vektoru  $v$ .

$$H(v) = H(v, \mathbf{0})$$

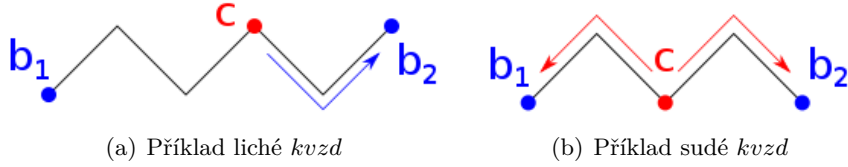
**Definice 13** *Kódová vzdálenost:* Kódová vzdálenost kódu  $\mathcal{K}$  je *minimální* Hammingova vzdálenost mezi všemi kódovými slovy.

$$kvzd(\mathcal{K}) = \min_{\substack{\forall b_1, b_2 \in \mathcal{B} \\ b_1 \neq b_2}} vzd(b_1, b_2)$$

Dále budeme značit  $d = kvzd(\mathcal{K})$ . Pokud je  $d > 1$ , tak je jasné, že můžeme za jistých okolností odhalit (detekovat), že při přenosu kódového slova nastala chyba. Pokud by ale nastalo  $d$  a více chyb, je možné, aby se z jednoho *kódového slova* stalo *kódové slovo* jiné (viz příklad s *Hammingovými* kódy v kapitole B.2.1).

### Detekční kód

Kód, který dokáže při dekódování zjistit, že při přenosu nastala chyba nazýváme kódem *detekčním*. Při kódové vzdálenosti  $d$  je z principu možné detekovat  $d - 1$  chyb.



Obrázek B.2: Ilustrace problému nalezení nejbližšího kódového slova

### Samoopravný kód

Kód, který dokáže při dekódování dokáže opravit chybu (způsobenou přenosem), nazýváme kódem *samoopravným*. Při kódové vzdálenosti  $d$  je z principu možné opravit  $t = \lfloor \frac{d-1}{2} \rfloor$  chyb. Operace *dekódování* potom z nekódového slova dokáže nalézt nejbližší (ve smyslu *Hammingově vzdálenosti*) slovo *kódové*. U samoopravných kódů uvádíme parametry včetně počtu chyb, které kód dokáže opravit, tedy  $(n, k, t)$ <sup>30</sup>.

Počet opravitelných chyb naznačují obrázky B.2. Vrcholy úseček představují vektory délky  $n$  mezi dvěma kódovými slovy  $b_1$  a  $b_2$  (naznačení nejkratší *kódové vzdálenosti* v prostoru  $GF(2^n)$ ). V případě, že  $d$  je liché (obrázek B.2(a)), vždy existuje jednoznačný nejbližší vektor. V případě, že  $d$  je sudé (obrázek B.2(b)), tak pokud přijatý vektor  $c$  leží přesně uprostřed mezi dvěma nejbližšími vektory, není možné rozhodnout, na které kódové slovo by se měl vektor  $c$  dekódovat.

Existuje několik kategorií *samoopravných* kódů. Definice kategorie kódu ve své podstatě určuje, jakým způsobem bude probíhat konstrukce kódu (respektive kódového slova), aby se zajistila určitá kódová vzdálenost  $d$  a při dekódování bylo možné nalézt patřičný počet chyb  $t$ .

## B.2 Lineární kódy

Dalším důležitým pojmem, který budeme používat jsou *lineární kódy*.

**Definice 14** *Lineární kód:* Nechť je zobrazení odpovídající kódu  $\mathcal{K}$  lineární, pak nazýváme tento kód lineárním.

Jinými slovy *kódová slova* kódu  $\mathcal{K}$  tvoří *lineární prostor* – přesněji *lineární podprostor* vektorového prostoru  $GF(2)^n$ . V tomto prostoru definujeme klasické operace sčítání dvou vektorů jako operaci *XOR* a násobení skaláru s vektorem jako operaci násobení po jednotlivých složkách vektoru. Je jasné, že v případě násobení skalárem 0 je výsledek operace násobení skalárem *nulový vektor* ( $0 \cdot \mathbf{v} = \mathbf{0}$ ) a násobením skalárem 1 získáme původní (nezměněný) vektor ( $1 \cdot \mathbf{v} = \mathbf{v}$ ).

<sup>30</sup> V některých zdrojích se místo počtu opravitelných chyb objevuje kódová vzdálenost, tedy  $(n, k, d)$ , což odpovídá  $(n, k, 2t + 1)$ .

Z definice lineárního prostoru též plyne, že *nulový vektor*  $\mathbf{0}$  je vždy kódovým slovem *lineárního* kódu  $\mathcal{K}$ .

**Tvrzení 3** *Kódová vzdálenost lineárního kódu  $\mathcal{K}$  odpovídá minimální váze ze všech kódových slov (kromě nulového vektoru).*

$$kvzd(\mathcal{K}) = \min_{\forall b \in \mathcal{B} \setminus \mathbf{0}} H(b)$$

### Náznak důkazu

Důkaz vyplývá z faktu, že minimální *vzdálenost* daného kódového slova  $b$  ke všem ostatním kódovým slovům je pro všechna kódová slova stejná:

$$\forall b : \min_{\forall b_i \in \mathcal{B} \setminus b} H(b, b_i) = \min_{\forall b_i \in \mathcal{B} \setminus b} H(b - b_i, 0) = \min_{\forall b_i \in \mathcal{B} \setminus b} H(b_i, 0)$$

sečtením (odečtením) dvou kódových slov vznikne opět kódové slovo. Proto  $b_j$  je pouze substituce naznačující *nějaké* kódové slovo. Pokud je pro všechny stejná, tak odpovídá *kódové vzdálenosti*. Když se tedy podíváme na *nulový vektor* (kódové slovo), tak nejbližší kódové slovo odpovídá kódovému slovu s nejnižší *Hammingovou vahou*.

**Definice 15** *Generující matice: Necht' soubor vektorů  $g_1, g_2, \dots, g_k$  tvoří bázi prostoru kódových slov  $\mathcal{B}$  lineárního kódu  $\mathcal{K}$ . Potom maticí  $G$ , sestavenou po řádcích vektory  $g_i$ , nazveme generující maticí kódu  $\mathcal{K}$ .*

Matice  $G$  je vlastně matice *lineárního zobrazení*  $\mathcal{K}$  z prostoru (všech) zpráv do prostoru kódových slov ( $\mathcal{A} \rightarrow \mathcal{B}$ <sup>31</sup>).

Operace zakódování  $K$  zprávy  $a$  potom u *lineárního* kódu odpovídá násobení vektoru s generující maticí:

$$K_{\mathcal{K}}(a) : b = aG$$

Toto maticové násobení odpovídá sečtení řádků matice  $G$ , které jsou určeny vektorem  $a$  (sečtení vektorů  $g_i$ ).

**Definice 16** *Systematický kód: Pokud je generující matice kódu  $\mathcal{K}$  ve tvaru  $G = (\mathbb{I}_k | F)$ , kde  $\mathbb{I}_k$  je jednotková matice  $k \times k$  a  $F$  je matice  $k \times r$ , říkáme, že kód  $\mathcal{K}$  je systematický.*

Prvních  $k$  bitů *kódových* slov *systematického* kódu pak přesně odpovídá původní zprávě  $a$ . Těmto bitům říkáme *informační* bity a posledním  $r$  bitům pak bity *kontrolní*. Při dekódování kódového slova pak stačí jednoduše odstranit *kontrolní* bity a zůstanou tak bity původní zprávy

$$D(c) : d = MSB_k(c)$$

Samozřejmě toto je možné pouze pokud bylo přijaté slovo *kódové*. Pro detekci a opravu chyb budeme potřebovat *kontrolní* matici.

---

<sup>31</sup> Kde  $\mathcal{A} = GF(2)^k$  a  $\mathcal{B} \subset GF(2)^n$ .



**Definice 17** *Kontrolní matice:*<sup>32</sup> Nechť  $G$  je generující matice lineárního kódu  $\mathcal{K}$ . Pak definujeme kontrolní matici  $H$  tohoto kódu jako:

$$GH^T = \mathbf{0}$$

Kde  $\mathbf{0}$  je nulová matice .

Vezmeme-li řádky matice  $H$  jako soubor vektorů  $h_i$ , pak jsou tyto vektory bází ortogonálního doplňku<sup>33</sup>  $\mathcal{H}$  k prostoru kódových slov  $\mathcal{B}$ . Neboli

$$\forall b \in \mathcal{B}, \forall h \in \mathcal{H} : b \perp h$$

Na matici  $H$  se lze dívat též jako na generující matici kódu  $\mathcal{K}'$  s parametry  $(n, n - k)$ , pak samozřejmě platí, že matice  $G$  je kontrolní maticí tohoto kódu. Kód  $\mathcal{K}'$  se nazývá *duálním kódem* ke kódu  $\mathcal{K}$ .

**Tvrzení 4** *Pokud je generující matice  $G$  v systematické formě  $G = (\mathbb{I}_k | F)$ , tak má kontrolní matice tvar*

$$H = (F^T | \mathbb{I}_r)$$

#### Důkaz

Dosadíme-li do definice kontrolní matice:

$$GH^T = (\mathbb{I}_k | F)(F^T | \mathbb{I}_r)^T = (\mathbb{I}_k | F)\begin{pmatrix} F \\ \mathbb{I}_r \end{pmatrix} = F + F = \mathbf{0}$$

tak je vidět, že matice  $H$  tuto definici splňuje.

Pokud  $G$  není v tomto systematickém tvaru, lze ji pomocí elementárních operací převést na matici  $G'$  v systematickém tvaru a získat dle tvrzení výše kontrolní matici  $H'$ . Matice  $H'$  je pak i kontrolní maticí k původní matici  $G$ , jelikož elementární úpravy nemění prostor, který matice generuje [2].

Tento způsob převodu matic je invertibilní a je tak možné získat generující matici z matice kontrolní. Lineární kód je tedy určen jednoznačně jak generující tak i kontrolní maticí.

**Definice 18** *Syndrom:* Nechť  $H$  je kontrolní matice lineárního kódu  $\mathcal{K}$  a  $c$  je přijatý vektor. Syndrom  $s$  tohoto přijatého vektoru je

$$s = cH^T$$

**Tvrzení 5** *Syndrom závisí pouze na chybovém vektoru  $e$  a pokud je  $e$  nulový vektor (pro kódová slova  $c$ ) je syndrom také nulový.*

<sup>32</sup> V některých zdrojích uváděna jako matice prověřková.

<sup>33</sup> Nebo též nulového prostoru.

**Důkaz**

Při dosazení  $c = b + e$  získáme rovnost:

$$s = cH^T = (b + e)H^T = bH^T + eH^T$$

a z definice *ortogonálního doplňku* platí:  $bH^T = \mathbf{0}$

$$\Rightarrow s = eH^T$$

Vypočítaný *syndrom* se používá pro detekci, zda bylo přijaté slovo *kódové* či nikoliv. *Samoopravné* kódy zpravidla využívají *syndrom* pro rekonstrukci chyby a opravení přijatého vektoru  $c$  na slovo kódové.

**B.2.1 Hammingovy kódy**

*Hammingovy* kódy jsou příkladem *lineárních samoopravných* kódů. Dokáží opravit *jednu chybu* a jejich parametry  $(n, k, t)$  jsou určeny de facto jedním parametrem  $r$ .

Pro každé  $r \geq 2$  můžeme sestavit *kontrolní matici Hammingova kódu* s parametry  $(n, k, t) = (2^r - 1, n - r, 1)$  jednoduše tak, že vygenerujeme všechny možné *nenulové* a vzájemně různé sloupcové vektory  $h_i$  (délky  $r$ ).

$$H = \begin{pmatrix} h_1 & h_2 & \dots & h_n \end{pmatrix}, h_i \in GF(2)^r \setminus \mathbf{0}, h_i \neq h_j$$

Pokud chceme získat *systematický* kód, tak  $k$  posledních sloupců bude tvořit *jednotkovou matici*  $\mathbb{I}_k$  a *generující matici* takového kódu získáme převodem z matice  $H$  popsaným výše.

**Oprava jedné chyby**

*Syndrom* délky  $r$  přijatého slova  $c$  vypočítáme výše definovaným způsobem

$$s = cH^T$$

Dle tvrzení výše víme, že *syndrom* závisí pouze na *chybovém* vektoru  $e$  (délky  $n$ ) – platí tedy, že  $s = eH^T$ .

V případě, že  $c$  je kódové slovo, bude *syndrom* nulový a z  $c$  tak můžeme rovnou *dekódovat* slovo  $d$  (vybráním *informačních bitů*).

Nyní předpokládejme, že nastala pouze 1 chyba v dimenzi  $i$ . Chybový vektor označíme  $e_i$ . Výpočtem  $e_i H^T$  tak získáme *syndrom*, který odpovídá  $i$ -tému sloupcovému vektoru matice  $H$  ( $h_i$ ), protože každý sloupec matice  $H$  je jiný a žádný není nulový. Dle *syndromu* jsme tedy schopni odhalit chybu  $e_i$  a z přijatého slova  $c$  získat  $c' = c + e_i$ . Slovo  $d$  pak dekódujeme stejným způsobem jako v případě bez chyby, ale z vektoru  $c'$ .

**Poznámka:** Pokud by nastaly chyby ve dvou dimenzích  $i$  a  $j$ , *syndrom* by tak byl součtem dvou různých sloupců matice  $H$  a vznikl by tak *syndrom* odpovídající úplně jinému sloupcovému vektoru  $(h_k, k \neq i \neq j)$ . Pokud by nastalo tři a více chyb, může se dokonce při přenosu stát z kódového slova  $b$  jiné kódové slovo. To plyne z faktu, že *kódová vzdálenost Hammingových kódů*  $d = 3$  [2].

### Příklad

Zvolme parametr  $r = 3$ . Potom  $n = 2^3 - 1 = 7$  a  $k = 7 - 3$ . Vygenerujeme tedy *kontrolní matici  $H$  Hammingova kódu* s parametry  $(7, 4, 1)$ :

$$H = \begin{pmatrix} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{pmatrix} = (F | \mathbb{I}_3)$$

Matice  $H$  je v systematickém tvaru (tři poslední sloupce tvoří jednotková matice) a můžeme ji tak snadno převést na *generující matici  $G$* :

$$G = (\mathbb{I}_4 | F^T) = \begin{pmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{pmatrix}$$

Zvolme  $a = (1010)$ . *Kódové slovo  $b$*  získáme z definice vynásobením vektoru  $a$  s maticí  $G$ , neboli sečteme 1. a 3. řádek matice  $G$ .

$$b = aG = (1010100)$$

Vyšleme tedy tento vektor kanálem příjemci.

1. V prvním případě nenastala žádná chyba. Vektor  $e$  je nulový a  $c$  tedy odpovídá  $b$ . Vypočteme *syndrom*

$$s_1 = cH^T = bH^T = (000)$$

Syndrom je nulový a víme tedy, že přijaté  $c$  je kódové slovo. Jelikož je kód *systematický*, dekódování  $d$  je realizováno výběrem prvních  $k$  dimenzí:

$$d = D(c) = MSB_4(1010100) = (1010)$$

Je vidět, že  $d$  odpovídá původní zprávě  $a$ .

2. V druhém případě nastala právě jedna chyba – vektor  $e = (0010000)$ . Přijatý vektor je nyní  $c = b + e = (1000100)$ . Vypočteme syndrom  $s_2$ :

$$s_2 = cH^T = (011)$$

Tento syndrom odpovídá 3. sloupci matice  $H$ . Invertujeme tedy 3. bit přijatého slova  $c$  a opět dekódujeme  $d$ :

$$d = D(c') = MSB_4(1010100) = 1010$$

Pokud nastala 1 chyba, byli jsme ji schopni opravit a získat původní zprávu  $a$ .

3. Ve třetím případě nastane chyba ve dvou dimenzích. Chybový vektor bude nyní  $e = (1000010)$  a přijatý vektor  $c = b + e = (0010110)$ . Vypočteme *syndrom*

$$s_3 = cH^T = (101)$$

Tento syndrom odpovídá 2. sloupci matice a tak při dekódování invertujeme 2. bit vektoru  $c$

$$d = D(c') = MSB_4(0110110) = 0110$$

Nyní je tedy dekódováním získáno slovo  $d$ , které neodpovídá původní zprávě  $a$

**Poznámka:** *Hammingovy kódy* jsou tzv. *perfektní kódy*. U *perfektního kódu* každý syndrom odpovídá nějaké (v tomto případě jedné) či žádné chybě. Pokud je *Hammingův* kód použitý pro opravu jedné chyby, tak pokud nastane 2 a více chyb, nebude dekódované slovo odpovídat původnímu a ani není možné tuto situaci nijak *detekovat*.

## Seznam použitých zkratk

- AG-kódy** *Algebraicko-Geometrické* (alias *Goppa*) kódy
- BCH** *Bose-Chaudhuri-Hocquenghem* kódy
- CPA** *Chosen Plaintext Attack* – útok s voleným otevřeným textem
- CCA** (**CCA1**) *Chosen Ciphertext Attack* – útok s voleným šifrovým textem
- CCA2** *Adaptive Chosen Ciphertext Attack* – útok s adaptivní volbou šifro-  
vého textu
- DH** Algoritmus *Diffie-Hellman*
- DSA** *Digital Signature Algorithm*
- ECC** *Elliptic Curve Cryptography*
- EEA** *Extended Euclidean Algorithm* – rozšířený Euklidův algoritmus
- GCD** *Greatest Common Divisor* – největší společný dělitel
- GRS** *Generalised Reed-Solomon code* – zobecněný Reed-Solomon kód
- GF** *Galois Field* – konečné těleso
- LSB** *Least Significant Bit/Byte* – nejméně významný bit/bajt
- MDPC** *Moderate Density Parity-Check* kódy
- MSB** *Most Significant Bit/Byte* – nejvíce významný bit/bajt
- OAEP** *Optimal Asymmetric Encryption Padding* – schéma pro asymetrické  
šifrování
- PKC** *Public-Key Cryptography* – asymetrická kryptografie s veřejným klíčem

**QC-MDPC** *Quasi-Cyclic MDPC* kódy

**QD** *Quasi-Dyadic – kvazi-dyadické – Goppa* kódy

**RSA** Algoritmus *RSA – Rivest, Shamir, Adleman*

**S&M** Algoritmus *Square-and-Multiply*

## Obsah přiloženého disku

README.md .....	stručný popis obsahu disku
implementace .....	adresář s implementací v softwaru <i>Mathematica</i>
_ src .....	adresář s připravenými <i>balíky</i>
_ *.nb .....	ukázky použití kódu
materials .....	adresář s dodatečnými soubory a skripty
mereni .....	adresář se soubory z měření
text .....	text práce
_ DP_Myslivec_Vojtech_2016.tex .....	kód textu ve formátu $\text{\LaTeX}$
_ DP_Myslivec_Vojtech_2016.pdf .....	text ve formátu pdf