ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

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 klíčů. 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 Tvrdík, 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.

2. května 2016

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 2. května 2016 .....

Odkaz na tuto práci

Myslivec, Vojtěch. *Asymetrický šifrovací algoritmus McEliece.* Diplomová práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2016.

Abstrakt

Tady bude nejaky kuuul abstakt

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

Abstract

Sem doplňte ekvivalent abstraktu Vaší práce v angličtině.

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

Obsah

Seznam obrázků

Seznam tabulek

Seznam algoritmů

Úvod

Tato práce se zabývá asymetrickým kryptosystémem *McEliece*. Mezi největší přednosti tohoto systému patří jeho odolnost vůči kvantovým počítačům a je tak jedním z vhodných kandidátů pro asymetrickou kryptografii pro postkvantovou dobu.

V prvních kapitolách této práce jsou popsány nezbytné primitivy z oblasti matematiky a teorie kódování, které jsou potřeba pro pochopení a použití kryptosystému McEliece. Jedná se především o počítání s *konečnými tělesy* a *polynomy* (kapitola 1) a binární *Goppa* kódy (kapitola 2).

Kryptosystému *McEliece* se věnuje kapitola 3. Kromě základního popisu generování klíčů a algoritmů pro šifrování a dešifrování je probráno i kryptosystém *Niederreiter* – „úprava“ kryptosystému *McEliece* pro získání *digitálního podpisu*. Jsou ukázány slabiny, nevýhody i možné útoky na kryptosystém McEliece a též zmíněna praktická varianta systému odolná vůči těmto aspektům.

V poslední části práce je probrána implementace kryptosystému *McEliece* v softwaru *Wolfram Mathematica* včetně změřených časových složitostí (kapitola 4),.

Kapitola 1

Obecná algebra

V této kapitole 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 [31, 43, 44] a tuto literaturu zároveň doporučujeme pro hlubší studium této problematiky.

Poznámka: Algoritmy zmíněné v následujících kapitolách jsou detailně – včetně pseudokódu – popsány v kapitole 4, která se zabývá konkrétní implementací algoritmů a operací.

1.1 Základní termíny

Pro ujasnění je uvedena definice tělesa:

Definice 1 (Těleso) *Nechť M je neprázdná množina $a + a \cdot$ binární operace*[1]*. Struktura $T = (M, +, \cdot)$ se nazývá těleso, pokud platí*

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

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

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 $n \in \mathbb{N} \wedge n \geq 1$.*

[1] Pro zjednodušení zápisu je $\cdot$ často vynecháváno.
[2] 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 *Gallois field*, dle francouzského matematika *Évariste Galois*).

1.2 Reprezentace prvků

Jak ukážeme dále, je vhodné prvky konečného tělesa $GF(p^n)$ reprezentovat jako *polynomy* s koeficienty z množiny $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$, 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 se je možné dívat též jako na vektor či n-tici koeficientů a_i:

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

V této práci budeme mezi těmito reprezentacemi nadále volně přechází, jak bude v daném kontextu potřeba[3].

1.3 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 4.

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

1.3.1 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 $GF(p)$:

$$A(x) + B(x) = \sum a_i x^i + \sum b_i x^i = \sum |a_i + b_i|_p \, x^i$$

[3] V některých materiálech se používá i obráceného zápisu $(a_0 a_1 \ldots a_{n-1})$.

1.3.2 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 $A(x) \mod P(x)$. Dále je třeba pro jednoznačné určení tělesa $GF(p^n)$ určit příslušný *ireducibilní* polynom, který bude použitý při operaci násobení.

Definice 2 *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:

$$A(x) \cdot B(x) = \sum a_i x^i \cdot \sum b_i x^i = \sum \left| \sum_{j+k=i} a_j \cdot b_k \right|_p x^i \mod P(x)$$

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*.

1.3.3 Umocňování

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

Definice 3 *Nechť a je prvkem (libovolného) tělesa T a číslo $n \in \mathbb{N}$. Operace umocňování definujeme následovně:*

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

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.

1.3.4 Inverze

Inverzi v *grupě* lze obecně definovat následovně:

Definice 4 (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.

$$\left|A(x) \cdot A(x)^{-1}\right|_p = \left|\frac{A(x)}{A(x)}\right|_p = 1$$

Tuto *multiplikativní inverzi* je třeba počítat *rozšířeným Euklidovým algoritmem pro polynomy* (*EEA*), či případně jinými algoritmy, jako je například *algoritmus Itoh-Teechai-Tsujii* (*ITT*) [46, 40].

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[4], nicméně v binárních tělesech lze toto dělení nahradit prostým posouváním a „odečítáním“ (respektive sčítáním) prvků.

[4] Někdy uváděno jako dlouhé dělení.

1.4 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 5 *Rozšíření tělesa: Nechť T je těleso a P podmnožina množiny tělesa 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 6 *Okruh polynomů: Nechť $\mathbb{F}$ 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 1 *Nechť $\mathbb{F}$ 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 $\mathbb{F}^n$.[5]

V této práci budeme nadále používat termín *rozšířené těleso* ve smyslu konečného tělesa reprezentované polynomy s koeficienty z tělesa $GF(2^m)$. Rozšířením tak získáme těleso $GF(2^m)^n$.[6]

Poznámka: Pokud rozšířením konečného tělesa vnikne 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.

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

[6] Toto těleso je izomorfní s tělesem $GF(2^{mn})$.

Kapitola 2

Teorie kódování

V této kapitole definujeme a vysvětlíme pojmy z teorie kódování, které jsou použité v kryptosystému *McEliece* (kapitola 3). Definice byly čerpané z [45, 2] a pro další studium této problematiky je též doporučeno [26].

V podkapitolách uvádíme základní značení a termíny z oblasti samoopravných (2.1) a lineárních (2.2) kódů. V poslední kapitole 2.3 se věnujeme *binárním Goppa kódům*, které jsou přímo použité v kryptosystému *McEliece*.

2.1 Samoopravné kódy

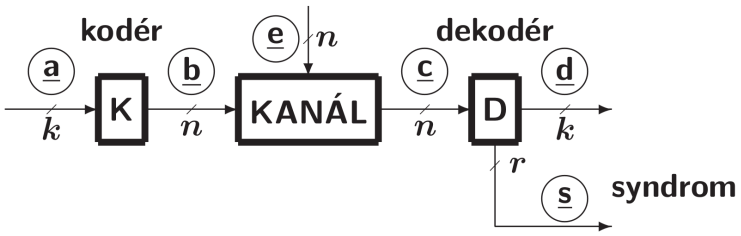
Teorie kódování spadá do oblasti teorie informace a zabývá se způsoby *zakódování* či *reprezentací* zpráv a jejich přenosem.

V této kapitole budeme používat následujícího značení (odpovídá obrázku 2.1):

$+$	binární bitová operace *XOR*
$-$	inverzní operace k $+$, tedy též *XOR*
a	zpráva délky k
$K()$	operace *zakódování*
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 ($c = b + e$)
$D()$	operace dekódování
d	*dekódovaná* zpráva; $d = D(c)$
s	*syndrom* přijaté zprávy/chyby (viz dále) délky r; zpravidla platí $r = n - k$

Definice 7 *(Binární) kód: Nechť existuje (prosté) zobrazení $\mathcal{K}$ z množiny všech možných zpráv a délky k do množiny kódových slov b délky n ($GF(2)^k \to GF(2)^n$). Pak toto zobrazení nazveme kódem $\mathcal{K}$ s parametry (n, k).*[7]

[7] Obecně lze *kód* definovat jako zobrazení $\mathcal{L}^k \to \mathcal{M}^n$, kde $\mathcal{L}$ je *abeceda* zpráv délky k a $\mathcal{M}$ abeceda kódových slov délky n.



Obrázek 2.1: Použité značení při kódování [45]

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

Z definice 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). Operací *zakódování* budeme rozumět aplikaci zobrazení $\mathcal{K}$ a operací dekódování aplikaci $\mathcal{K}^{-1}$, tedy získání původní zprávy z (*kódového*) slova.

Definice 8 *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*:* $vzd(u, v) = \sum |u_i - v_i|$

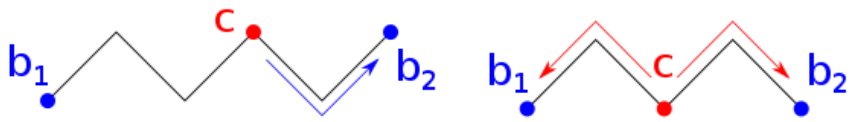
Hammingovu váhu vektoru v pak definujeme jako *Hammingovu vzdálenost* vektoru v od *nulového vektoru* patřičné délky. Jinými slovy je to počet *nenulových* bitů („jedniček“) vektoru v.

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

Definice 9 *Kódová vzdálenost: Kódová vzdálenost kódu* $\mathcal{K}$ *je minimální* Hammingova vzdálenost *mezi všemi kódovými slovy.*

$$kvzd(\mathcal{K}) = \min_{\substack{\forall b1, b2 \in \mathcal{B} \\ b1 \neq b2}} vzd(b1, b2)$$

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 2.2.1).



(a) Příklad liché *kvzd* (b) Příklad sudé *kvzd*

Obrázek 2.2: Ilustrace problému nalezení nejbližšího kódového slova

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.

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)[8].

Počet opravitelných chyb naznačují obrázky 2.1. 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 sudé 2.2(b), vždy existuje jednoznačný nejbližší vektor. V případě, že d je liché 2.2(a), 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.

V této práci budeme nadále pracovat pouze se *samoopravnými kódy*.

2.2 Lineární kódy

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

Definice 10 *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

[8] V některých zdrojích se místo počtu opravitelných chyb objevuje kódová vzdálenost, tedy (n, k, d), což odpovídá $(n, k, 2t+1)$.

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 ($1 \cdot \mathbf{v} = \mathbf{v}$).

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

Tvrzení 2 *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:

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

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 11 *Generující matice: Nechť soubor vektorů $g_1, g_2, \ldots, g_k$ tvoří bázi prostoru kódových slov $\mathcal{B}$* lineárního kódu *$\mathcal{K}$. Potom matici G, sestavenou po řádkách 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 ($\mathcal{A} \to \mathcal{B}$ [9]).

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

$$K_G(a) : b = aG$$

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

Definice 12 *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 $r \times k$, říkáme, že kód $\mathcal{K}$ je* systematický*.*

[9] Kde $\mathcal{A} = GF(2)^k$ a $\mathcal{B} \subset\subset GF(2)^n$.

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.

Definice 13 *Kontrolní matice:*[10] *Nechť G je* generující *matice* lineárního kódu $\mathcal{K}$. *Pak definujeme* kontrolní matici H *tohoto kódu jako:*

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

Kde $\boldsymbol{0}$ je nulová matice .

Vezmeme-li řádky matice H jako soubor vektorů h_i, pak jsou tyto vektory *bází ortogonálního doplňku*[11] $\mathcal{H}$ 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 3 *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:

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

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.

[10] V některých zdrojích uváděna jako matice *prověrková*.
[11] Nebo též *nulového* prostoru.

Definice 14 *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 4 Syndrom *záleží pouze na chybovém vektoru e a pokud je e nulový vektor (pro kódová slova c) je syndrom také* nulový.

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.

2.2.1 Hammingovy kódy

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

Pro každé $r \geq 2$ můžeme sestrojit *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 $(7, 4, 1)$:

$$H = \left(\begin{array}{cccc|ccc} 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right) = (F|\mathbb{I}_3)$$

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

$$G = \left(\mathbb{I}_4|F^T\right) = \left(\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{array}\right)$$

Zvolme $a = (1010)$. *Kódové* slovo b získáme z definice vynásobením vektoru s maticí $b = aG$, 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 tedy nulový a $c = b$. Vypočteme *syndrom*

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

   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 případe 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*.

2.3 Binární Goppa kódy

Novou kategorii *lineárních* kódů definoval v roce 1970 *Valery Goppa* v [18]. 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 [26].

Poznámka: Obecné *Goppa* kódy jsou definovány pomocí *algebraických křivek*[12], nicméně v této práci se budeme zabývat pouze podkategorií, tzv. *binárními Goppa kódy*.

[12] *Goppa* kódy jsou též nazvány jako *algebraické geometrické* (AG) kódy.

2.3.1 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.

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

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

Pak *binární Goppa* kód (prostor kódových slov) Γ definujeme:

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

Polynom $g(x)$ nazýváme *Goppův* polynom a n-tici L *podporou* kódu[13].

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

Poznámka: U *binárních Goppa* kódů je polynom $x - L_i$ je vlastně prvek $(0 \ldots 01)(L_i)$. Důvod podmínky $g(L_i) \neq \mathbf{0}$ 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.

Ireducibilní binární Goppa kódy

Pokud je g *ireducibilní*, nazveme Γ *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 $\mathbf{0}$) platí podmínka $g(L_i) \neq \mathbf{0}$. Takový kód má tedy paramtery $(n, k, t) = (2^m, 2^m - mt, t)$, a jsou tedy jednoznačně určené parametrem m („velikostí vnitřního tělesa“) a stupněm polynomu g, neboli počtem opravitelných chyb t.

Sestrojení kontrolní matice

Z definice lze sestrojit *kontrolní* matici H v následujícím tvaru (detailní postup sestrojení matice lze nalézt např. v [22]):

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

[13] Anglicky *Goppa polynomial* g a *support* L.

a tato matice 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 sestrojit jednodušší kontrolní matici:

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

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

H je $n \times t$ matice nad tělesem $GF(2^m)$. H nad $GF(2)$ získáme jednoduše „rozbalením“ prvků $GF(2^m)$ do sloupcových vektorů m bitů. *Binární kontrolní* matice H pak má rozměry $n \times r = (2^m) \times (mt)$ a *generující* matici G získáme klasickým převodem, jak bylo popsáno v kapitole 2.2.

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 [33]. Další algoritmy pro dekódování algoritmů – především tzv. *List Decoding* algoritmy – se dají nalézt v [35, 7].

Pattersonův algoritmus

Syndrom přijatého slova c je možné počítat z kontrolní matice H nebo též jako polynom z definice kódu Γ:

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

Takto spočítaný syndrom jistě závisí pouze na chybovém vektoru e:

$$\sum_{i=1}^{n} \frac{c_i}{x - L_i} = \sum_{i=1}^{n} \frac{b_i}{x - L_i} + \sum_{i=1}^{n} \frac{e_i}{x - L_i}$$

$$\sum_{i=1}^{n} \frac{b_i}{x - L_i} + \sum_{i=1}^{n} \frac{e_i}{x - L_i} \equiv \sum_{i=1}^{n} \frac{e_i}{x - L_i} \mod g(x)$$

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 α stupně $\leq \lfloor \frac{t}{2} \rfloor$ a β stupně $\leq \lfloor \frac{t-1}{2} \rfloor$ tak, že:

$$\alpha(x) \equiv \beta(x) r(x) \mod g(x)$$

3. Sestrojíme polynom $\sigma = \alpha^2 + x\beta^2$, tzv. *lokátor chyb*.
4. Kořeny L_i (z podpory L) polynomu σ 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 [33]. Poznámky k jednotlivým krokům algoritmu uvádíme níže.

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

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

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

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

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 α a β je ve skutečnosti snadné. Rovnice $\alpha(x) \equiv \beta(x)r(x) \mod g(x)$ je rovnicí, která vzniká při výpočtu *rozšířeného Euklidova algoritmu*. Polynom α odpovídá „zbytku“ a β „koeficientu“ při výpočtu *EEA* (viz příklad *EEA* v kapitole 4.1.3.4). 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 asymptoticky nejnáročnější. Základní způsob pro nalezení kořenů polynomu je hrubou silou vypočítat hodnotu $\sigma(L_i)$ pro všechny koeficienty L_i z podpory L.

Efektivnější algoritmem je tzv. *Chienův způsob* hledání kořenů[14] [45, 19], který využívá výpočtu kořenů pomocí primitivního prvku α. Pro všechny prvky α^i tělesa $GF(2^m)$ (kromě nulového prvku) platí:

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

Z posledního řádku je vidět, jak lze tohoto faktu při výpočtu všech kořenů využít. Vypočteme-li tedy hodnotu $\sigma(\alpha)$, tak další hodnoty $\sigma\left(\alpha^i\right)$ vypočteme pomocí $s-1$ operací násobení a $s-1$ sčítání $\Rightarrow O(t)$ násobení. Prostým dosazením všech prvků do polynomu σ musíme pro *každý* prvek vypočítat $\sim s-2$ mocnin navíc.

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

[14] Tento algoritmus byl původně navržený pro *BCH* kódy v [12]

Kapitola 3

Kryptosystém McEliece

Kryptosystém *McEliece* je asymetrický šifrovací algoritmus, publikovaný poprvé v roce *1978 Robertem McEliece* [1]. V podkapitole3.1 uvádíme algoritmy navržené *Robertem McEliece* z tohoto článku. Dále v 3.2 zmíníme příbuzný kryptosystém *Niederreiter* a v 3.3 schéma pro získání elektronického podpisu. Dále jsou probrány výsledky a závěry existujících kryptoanalýz systému *McEliece* (3.4) a nakonec se věnujeme aktuálním variantám a úpravám *kryptosystému* (3.5).

Poznámka: V této kapitole nadále předpokládáme počítání s hodnotami z tělesa $GF(2)$, respektive s *bity*.

3.1 Asymetrické šifrování McEliece

Asymetrický kryptosystém *McEliece* je založený na lineárních samoopravných kódech. V následujících odstavcích systém popsán tak, jak byl definován v [1]:

3.1.1 Generování klíčů

Generování klíčů probíhá následovně:

1. Zvolíme *lineární kód*[15] (n, k), opravující t chyb (a pro který je znám efektivní dekódovací algoritmus) s odpovídající $k \times n$ *generující maticí* G.
2. Vygenerujeme *náhodnou* $k \times k$ *regulární* matici S.
3. Vygenerujeme *náhodnou* $n \times n$ *permutační* matici P.
4. Vypočítáme $k \times n$ matici $\hat{G} = SGP$.

[15] V článku 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.

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

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 [19]. 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 [34]

3.1.2 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 důkaz dešifrování, neboli vysvětlení, že dešifrovacím algoritmem je 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[16].
2. Šifrovou zprávu c délky n sestrojíme následujícím způsobem:

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

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 $\hat{m}$
   $\hat{m} = Dek_G(\hat{c})$
3. Vypočítáme původní zpráva m: $m = \hat{m}S^{-1}$

[16] V původním článku je uvedeno maximálně t, nicméně v pozdějších pracích na toto téma se uvádí právě t. Důvody jsou vysvětleny v kapitole 3.4.

Důkaz dešifrování
Důkaz, že výsledkem dešifrování je opět původní zpráva je následující:

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

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

- 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).

$$Dek_G(\hat{c}) = \hat{m}$$

- V posledním kroku stačí opět dosadit výše použitou substituci:

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

Dešifrováním je tedy získána původní zpráva m.

3.1.3 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*.

3.1.3.1 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}.

3.1.3.2 Velikost klíčů

Největší nevýhodou *kryptosystému McEliece* je velikost klíčů. Již v původním článku jsou navrhovány parametry $n = 1024$, $k = 524$ a $t = 50$[17]. 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 *řídká* – každý *řádek* (respektive i *sloupec*) obsahuje pouze jednu jedničku, jinak je nulová. Je to permutační matice a lze

[17] Jak bude zmíněno dále, velikost těchto parametrů je pro dnešní použití nedostatečná.

tak uchovat ve formě $\log_2 n$ n-bitových indexů. Pro výše zmíněné hodnoty je to 10240 b = 10 kb.

Při použití *binárních Goppa kódů* s těmito parametry je potřeba k uložení informace o použitém kódu ≈ 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 3.5.1.

3.1.3.3 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 dekódování vektoru $\hat{m}$. Viz kapitola 3.4.2 a tabulka 3.3.

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 3.4.1.1).

3.1.4.1 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 jednotlivých matic je pro velká n a k obrovské, a získat tak původní matice hrubou silou je *neschůdné*[18].

3.1.4.2 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].

[18] Např. jen počet možných *permutačních matic* je $n!$. Počet *generujících* matic závisí na zvoleném kódu.

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

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

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

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

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.

3.2 Kryptosystém Niederreiter

V roce 1986 publikoval *Harald Niederreiter* v [30] 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*.

3.2.1 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 klíč*.

3.2.2 Algoritmy pro šifrování a dešifrování

V této podkapitole uvedeme algoritmy pro šifrování a dešifrování z [30] a důkaz toho, že dešifrovacím algoritmem je získána původní zpráva.

Šifrování

Šifrování zprávy probíhá následujícím způsobem:

1. Zpráva m dlouhá 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*[19] 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 dekódovacího 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 $\left(S^T\right)^{-1}$.

Důkaz dešifrování

Důkaz, že výsledkem dešifrování je opět původní zpráva je následující:

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

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

[19] 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 dekódovacího algoritmu získáme $\hat{m}$ jako *chybový vektor*.

- Nakonec se jen vynásobí inverzí matice P^T

3.2.3 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 dekódovat pouze za znalosti skrytého lineárního kódu.

V [42] 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 [30, 13].

3.3 Elektronický podpis

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

3.3.1 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 dekódovat 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 stejným způsobem [13].

3.3.2 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*.

3.3.2.1 Vyhovující parametry

V článku je dokázán vzorec pro pravděpodobnost, že náhodný *syndrom* délky $n - k$ (a při použití *Goppa kódů*) je možné dekódovat 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[20] 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 dekódovat jako *syndrom* je $\frac{1}{9!} \approx 2^{-19}$. Pro získání platného *syndromu* bude tedy nutné v průměru vygenerovat 2^{19} *vektorů*.

3.3.2.2 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 dekódovat, 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

[20] S ohledem na útok *Canteaut-Chabaud* [11].

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.

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.

3.3.3 Algoritmy schématu pro digitální podpis

Algoritmus pro podepisování
Podpis sestrojíme následujícím způsobem:

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

Algoritmus pro ověření
Ověření probíhá následujícím způsobem:

- Převedeme 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.

Poznámka: Bezpečnost schématu pro elektronický podpis závisí na jednosměrné funkci dekódování syndromu. Tuto operaci není možné provést bez znalosti *soukromého klíče* – matic H, S a P [30, 42].

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*).

3.4 Kryptoanalýza systému McEliece

Již v původním článku [1] byly naznačeny 2 aspekty, díky kterým je možné považovat kryptosystém McEliece *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 3.4.1.1. 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 3.4.2.

3.4.1 Útoky na McEliece

V této kapitole 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* [36]. 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 *šifrového 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 na zneužití několika *slabin*, které jsou probrány ve zvláštní kapitole 3.4.3[21].

3.4.1.1 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.

[21] Nejedná se totiž o útoky na kryptosystém ale spíše o nepříjemné *vlastnosti* kryptosystému, se kterými je nutné počítat.

V původním článku, kde byl definován kryptosystém *Niederreiter*, bylo navrženo použití *zobecněných Reed-Solomon* (*GRS*) kódů [30]. V [38] bylo prokázáno, že je možné skrytou strukturu *GRS* kódu odhalit v polynomiálním čase. Stejné 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, 41].

3.4.1.2 Support Splitting Algorithm

Tento algoritmus, navržený Nicolasem Sendrier, dokáže v *polynomiálním čase* (přibližně $O(n^4)$) určit, zda 2 lineární kódy jsou *permutačně ekvivalentní* [36].

Definice 15 *Nechť existují dva lineární kódy $\mathcal{K}_1$ a $\mathcal{K}_2$. Říkáme, že tyto kódy jsou* permutačně ekvivalentní*, pokud všechna kódová slova kódu $\mathcal{K}_1$ lze převést na kódová slova $\mathcal{K}_2$ 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* $\hat{G}$. 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 [35].

3.4.1.3 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 3.1.4. Později byl tento útok formalizován a zobecněn v [24].

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 $\hat{G}_{\mathcal{K}}$ byla *regulární* a bylo možné vyřešit vzniklou soustavu rovnic

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

Tomuto útoku brání fakt, že útočník neví, které bity šifrového 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 $c_{\mathcal{K}}$ 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 $e_{\mathcal{K}}$, 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 při rozumných velikostech kódů volit $j = 2$. Tento útok v době publikování snížil složitost útoku na *kryptosystém McEliece* přibližně $2^1$1-krát [24].

3.4.1.4 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 $\mathcal{K}'$ 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 [25, 39, 11] nejdříve hledají kódová slova v redukovaném 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* [25], *Stern* [39] a *Canteaut* a *Chabaud* [11] se liší hlavně ve způsobu výběru dimenzí S. Poslední z představených algoritmů dosahuje nejlepších výsledků.

3.4.1.5 Další útoky

Existují též návrhy dalších útoků jako jsou například statistické útoky [20] či útok založený na *bodových mřížích* [10]. Jako další zdroje pro zkoumání těchto útoků jsou doporučeny články [35, 15].

3.4.2 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 $O\left(2^n\right)$ 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) [31].

Kryptosystém *McEliece* má na rozdíl např. od *RSA* několik parametrů – n, k, t – a celkové množství variant je tedy velmi mnoho. 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 3.1 shrnuje parametry kryptosystému *McEliece* pro dosažení požadované míry bezpečnosti dle [6] a tabulka 3.2 dle [35]. Tyto tabulky obsahují informaci o velikosti *veřejného klíče* v *systematické formě*. Tabulka 3.3 inspirovaná z [15, 31] porovnává asymptotické složitosti šifrování a dešifrování kryptosystému *McEliece* a *RSA*.

Míra bezpečnosti původního navrženého kryptosystému $(1024, 524, 50)$ se dle [11, 35] pohybuje mezi 50-60 *bity bezpečnosti* a tyto parametry jsou tedy pro praktické použití nedostatečné.

3.4.3 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 šifrového textu – tzv. *CCA2* útok,

Těmto slabinám se dá vyhnout díky použití *CCA2* bezpečné konverzi šifrového textu, kterou popíšeme v kapitole 3.5.2.

3.4.3.1 Malleability

Použití šifrování tak, jak je definováno v kapitole 3.1.2 umožňuje deterministickým způsobem změnit (neznámou) zašifrovanou zprávu – tzv. *mealleability*.

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 3.1: 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 3.2: Míra bezpečnosti *McEliece* dle [35]

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

Tabulka 3.3: Porovnání *McEliece* a *RSA* dle [15, 31]

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 $m_1 + m_2$.

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

Podobnou slabinu mají i algoritmy *RSA* či *ElGamal* [47]. Stejně jako u těchto algoritmů (např. *OAEP* pro *RSA*) i pro *McEliece* se dá tomuto útoku efektivně bránit předem daným formátem zprávy a *paddingem*.

3.4.3.2 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 $\binom{n}{t}$[22] – 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 Δm 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.

[22] Pro praktické parametry kryptosystému platí $n \gg t$.

3.4.3.3 Znalost části otevřeného textu

Složitost útoku na šifrovanou zprávu lze též velmi zjednodušit, pokud útočník bude znát alespoň část otevřeného textu. Nechť množina $\mathcal{I} \subset \{1, 2, \ldots, k\}$ 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 $\mathcal{J}$.

3.4.3.4 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[23].

Útočník, který zachytí zašifrovanou zprávu c, k ni přičte vektor s *Hammingovou vahou* 1: $(0 \ldots 010 \ldots 0)$. 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 dekódovat. 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 dekódování) a pokud není rovna t, je jasné, že nastalo k manipulaci se šifrovým textem.

[23] Podobně jako např. útok *Paddding Oracle* u blokových šifer [47].

3.5 Moderní varianty a úpravy

Použití kryptosystému *McEliece* tak, jak byl popsán na začátku kapitoly 3 by bylo pro účely šifrování velmi nerozumné a nepraktické. To hlavně z důvodu slabin, kterými algoritmus trpí (kapitola 3.4.3) a velikosti klíčů, které jsou v základní variantě větší než je nezbytně nutné. V následujících kapitolách probereme několik úprav *kryptosystému* pro jeho praktické použití.

3.5.1 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 vyměnit původní *Goppa kódy* jinými, kompaktnějšími lineárními kódy, neúspěšné. Nalezly se slabiny ve struktuře kódu, které lze využít pro jejich sestrojení bez znalosti tajných matic S a P (viz kapitola 3.4.1.1. Jediné alternativní kódy, jejichž použití zatím nebylo prolomeno, jsou *kvazi-dyadické Goppa kódy*, které zmíníme v kapitole 3.5.1.2 a *MDPC* kódy v kapitole 3.5.1.3.

Kromě definovaného kódu jsou v *soukromém klíči* obsažené též dvě velké matice S a P. Snížením velikosti těchto matic se zabývá následující kapitola.

Veřejný klíč je pouze jedna matice – „zamaskovaná“ generující $n \times k$ 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 *jednotkové matici* 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 kryptosystému při použití takové matice, je nutné použít *CCA2-odolnou* konverzi (viz kapitola 3.5.2).

3.5.1.1 Význam matic S a P

Jak jsme již zmínili v kapitole 3.1.3.2, *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 [32] 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 žádnou bezpečnostní účel pro skrytí matice G. Naopak matice P je velmi důležitá a prozrazení této permutace by znamenalo prozrazení *soukromého klíče*.

3.5.1.2 Kvazi-dyadické Goppa kódy

Jako jedna z úspěšných metod na zkrácení klíčů se v posledních letech jeví použití *kvazi-dyadických Goppa kódů* [28].

Definice 16 *Dyadická matice:*

- *Každá* 1×1 *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 17 *Kvazi-dyadická matice: Matice, která není* dyadická*, ale skládá se z* dyadických *submatic je* kvazi-dyadická.

Dyadická matice 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 [28] autoři ukázali, že je možné sestrojit (binární) *Goppa kód*, který má kontrolní matici v *dyadické* formě – tzv. *dyadický Goppa kód*. Takto sestrojený kód by ale 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* [28]. Implementace *kryptosystému* s *kvazi-dyadickými Goppa* kódy lze nalézt např v [32, 23]

3.5.1.3 MDPC McEliece

Jedna z nejnovějších variant kryptosystému *McEliece* je použití *Moderate Density Parity-Check* (*MDPC*) kódů a kvazi-cyklických *MDPC* kódů. Autoři v [29] navrhli použití těchto kódů v roce 2013 a dokázali nalézt klíče o velikosti přibližně 4 kb (!), které odpovídají 80 bitům bezpečnosti.

3.5.2 CCA2-odolná konverze

V kapitole 3.4.3 jsme se zmínili, že základní varianta algoritmu *McEliece* trpí některými slabinami. Kvůli těmto slabinám by nebylo možné algoritmů 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 *volaným 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 [21] uvádí, že tyto konverze nejsou *CCA2*-odolné a tak stále tak umožňují např. *reakční útok* (viz kapitola 3.4.3). Sami pak navrhli 3 možné *CCA2*-odolné konverze, z nichž třetí – označená jako *Kobara-Imai* γ *konverze* – je nejúčinější. Tato konverze γ je popsána algoritmem níže a ilustrována obrázkem 3.1.

Značení

V následujícím algoritmu použijeme toto značení:

$(a\|b)$	zřetězení vektorů a a b
m	otevřený text
$const$	veřejně známá konstanta
r	náhodné číslo (*seed*)
$prep(m)$	funkce na doplnění zprávy na požadovanou délku (jednoznačný *padding*)
$hash(l)$	kryptograficky bezpečná *hashovací* funkce s výstupem délky $\log_2 \binom{n}{t}$ bitů
$rand(r)$	kryptograficky bezpečná funkce inicializovaná *seedem* r, která vrací (pseudonáhodný) vektor (*CSPRNG*)
$conv$	invertibilní konverze čísla $\leq \binom{n}{t}$ na odpovídající vektor délky n a váhy t (viz též kapitola 3.3.2)
$E_{\hat{G}}(m, e)$	šifrovací algoritmus *McEliece* (vstupem je zpráva m a chybový vektor z)
$D_G(c)$	dešifrovací algoritmus *McEliece*
$MSB_n(l)$	n nejvíce významných (levých) bitů vektoru l
$LSB_n(l)$	n nejméně významných (pravých) bitů vektoru l

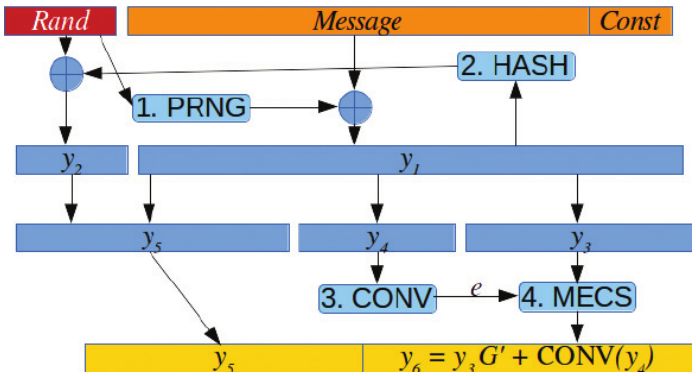
Délky vektorů

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	$n + \|const\| + \|r\| - \|y_4\| - \|y_3\|$

Algoritmus 1 Konverze Kobara-Imai γ

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

function DECRYPT(c, $const$)
  $y_5 \leftarrow MSB_{|c|-n}$
  $(y_3, e) \leftarrow D_G(LSB_n(c))$ ▷ 4.
  $y_4 \leftarrow conv^{-1}(e)$ ▷ 3.
  $(y_2|y_1) \leftarrow (y_5|y_4|y_3)$
  $r \leftarrow y_2 + hash(y_1)$ ▷ 2.
  $(\bar{m}|\bar{const}) \leftarrow y_1 + rand(r)$ ▷ 1.
  if $\bar{const} = const$ **then**
    return $prep^{-1}(\bar{m})$
  else
    return $NULL$ ▷ zamítni c
  end if
end function



Obrázek 3.1: Diagram CCA2-odolné konverze *Kobara-Imai* γ [35, 21]

3.5.3 Odolnost vůči kvantovým počítačům

Jeden z hlavních důvodů popularity algoritmu *McEliece* je fakt, není znám algoritmus pro kvantový počítač, který by dokázal kryptosystém prolomit rychleji, než na běžných počítačích [14]. Kryptosystém je tak zařazen mezi kandidáty asymetrické kryptografie pro tzv. *post-kvantovou* dobu [8] a jeho varianta s kvazi-cyklickými *MDPC* kódy se vyskytla v draftu z roku 2016 společnosti *IEEE* mezi doporučenými *post-kvantovými* asymetrickými kryptosystémy [37].

Kapitola 4

Implementace

Pro implementaci kryptosystému *McEliece* v této práci jsme zvolili software *Wolfram Mathematica* [48]. 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* –, kde každou z nich lze využít jako *balík* či *knihovnu* pro další výpočty. Následující kapitoly popisují jednotlivé části.

4.1 Binární konečná tělesa

V této podkapitole pojednáváme o implementaci *binárních konečných těles* včetně jejich *rozšíření*. Zmíníme existující řešení v softwaru *Mathematica*, zvolenou implementace a popíšeme implementované algoritmy.

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.

4.1.1 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.

4.1.1.1 Operace s polynomy

Software *Mathematica* obsahuje funkce pro operace s polynomy nad reálnými (případně i komplexními) čísly. Většina těchto funkcí má volitelnou *možnost*[24] *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*[25]. Tento požadavek je celkem nepraktický, protože definování této proměnné kdekoliv 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.

4.1.1.2 Balík `FiniteFields`

Balík *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 určené *seznamem*[26] 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*.

4.1.2 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:

[24] Anglicky se tento termín v softwaru *Mathematica* nazývá *Option*.
[25] Jinými slovy proměnná, která nemá definovanou hodnotu.
[26] *Seznamem* se myslí struktura v softwaru *Mathematica* – *List*.

- 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 dovolený.

- 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 nejdřív *operandy* a poté *modul*.
  Například pro prvky $a, b \in GF(p^{\cdots})$, $m \in \mathbb{N}$ a odpovídající *modul*:
  $krat[a, b, modul]$
  $inverze[a, modul]$
  $mocnina[a, m, modul]$
  ...

- 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 *polynomy* 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 *koeficientů*).

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*, ...).

4.1.3 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 a příkladů použití.

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

Zápis	Význam
`foo[bar]`	Volání funkce *foo* s argumentem *bar*
`ham[[i]]`	*i*-tý prvek seznamu (pole) *ham*

Tabulka 4.1: Prvky syntaxe jazyka softwaru *Mathematica*

4.1.3.1 Sčítání

Jelikož operace sčítání se v jakémkoliv *tělese* provádí po jednotlivých koeficientech *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 koeficienty 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(p^n), p je prvočíslo
2:     return Mod[a + b, p]
3: end function
```

```
1: function PLUS[ a, b, p ]                    ▷ Pro GF(q^n), q je p^···
2:     for i ← 1 ... Length[a] do
3:         c[[i]] ← plus[a[[i]], b[[i]], p]
4:     end for
5:     return c
6: end function
```

Poznámka: U dalších operací s prvky z tělesa $GF(p^n)$ (kde p je prvočíslo) se prvky (*seznamy*) převádějí na polynomy a využívá se implementovaných funkcí softwaru *Mathematica*. Z tohoto důvodu jsou nadále uváděné algoritmy pouze pro *rozšířená tělesa* $GF(q^n)$, kde q je nějaká mocnina prvočísla.

4.1.3.2 Redukce polynomu

Redukce polynomu (neboli *modulo* polynom) se používá ve většině dalších funkcí. Tato funkce se volá se dvěma parametry – prvkem a a polynomem

(*modulem*) m. Funkce vrátí zbytek polynomu a po dělení polynomem m.

Redukce polynomu pro *rozšířená tělesa* je inspirovaná *Comb metodou* z [27]. 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, $\{m, modul_{vnitrni}\}$]
2: $l_a \leftarrow stupen[a] + 1$ ▷ Délka redukovaného polynomu
3: $l_m \leftarrow stupen[m]$ ▷ Výsledná délka redukovaného polynomu
// Převedení m na *monický* polynom
4: $koef \leftarrow inverze[m[[1]], modul_{vnitrni}]$ ▷ Inverze nejvyššího koeficientu
5: $m \leftarrow krat[koef, m, modul_{vnitrni}]$ ▷ Násobení skalárem
6: $m \leftarrow PadRight[m, l_a - l_m]$ ▷ Natáhnutí polynomu na délku a
7: **for** $i \leftarrow 1 \ldots l_a - l_m$ **do**
8: $s \leftarrow krat[a[[i]], m, modul_{vnitrni}]$ ▷ Skalární násobek
9: $a \leftarrow plus[a, s, 2]$ ▷ Odečtení v binárním tělese
10: $m \leftarrow RotateRight[m]$ ▷ 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):

$$(10)(10)(00)(00)(00)(01) \mod (10)(01)(11)(10):$$

$$\begin{array}{r}
(10)\,(10)\,(00)\,(00)\,(00)\,(01) \\
\hline
(10)\,(01)\,(11)\,(10)\,(00)\,(00) \\
(00)\,(11)\,(10)\,(01)\,(11)\,(00) \\
(00)\,(00)\,(01)\,(11)\,(10)\,(01) \\
\hline
(00)\,(01)\,(00)
\end{array}$$

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

4.1.3.3 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 ← charakteristika[modul]                ▷ Charakteristika tělesa
       // Natažení na výslednou délku
3:     b ← PadLeft[b, stupen[a] + stupen[b] + 1]
4:     c ← nulovyPolynom[...]      ▷ Nulový polynom nad vnitřním tělesem

5:     for i ← stupen ... 1 do
6:         s ← krat[a[[i]], b, modul_vnitrni]          ▷ Skalární násobek
7:         c ← plus[c, s, p]
8:         b ← 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):

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

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

4.1.3.4 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) *koeficientu* polynomu p_i.

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

Poznámka: Pro výpočet dělení je v *rozšířených tělesech* potřeba vypočítat inverzi největšího koeficientu dělitele[27] 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	Koeficienty
	(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)

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

[27] Zde je patrná rekurzivní vlastnost tohoto algoritmu, kdy pro výpočet inverze prvku v tělese $GF(q^n)$ je třeba vypočítat inverzi v tělese $GF(q)$.

Algoritmus 5 Inverze prvků – *Rozšířený Euklidův algoritmus*

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

4.1.3.5 Druhá mocnina

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

Tvrzení 5 *Nechť* $A = (a_n \dots a_2 a_1 a_0)$ *je prvek tělesa s* charakteristikou 2*, potom platí:*

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

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 ← 1 ... Length[i] do
3:         a[[i]] ← naDruhou[a[[i]], modul_vnitrni]
4:     end for
5:     nula ← nulovyPolynom[...]          ▷ Odpovídající nulový koeficient
6:     a ← Riffle[a, nula]                ▷ Proloží koeficienty prvkem nula
7:     return redukujPolynom[a, modul]
8: end function
```

Náznak důkazu

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

4.1.3.6 Mocnění

Mocnění *polynomů* je implementováno pomocí algoritmu *Square-and-Multiply* (*SM*). Algoritmus využívá faktu, že libovolnou mocninu lze rozložit na součin mocnin čtverců (^{2}, ^{4}, ^{8}, ...). Konkrétně byla implementována varianta provádějící výpočet od nejvíce významného bitu exponentu[28]. Algoritmus má

[28] Uváděna jako *MSB* – z anglického *Most Significant Bit*.

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 $a^e \mod modul$ – *Square-and-Multiply*

```
1: function UMOCNI[ a, e, modul ]
2:     if e = 0 then
3:         return nulovyPolynom[...]                ▷ Nulový prvek tělesa
4:     end if
5:     rozvoj ← IntegerDigits[e, 2]                 ▷ Binární rozvoj exponentu
6:     c ← a                                        ▷ rozvoj[[1]] je vždy 1
7:     for i ← 2 ... Length[rozvoj] do
8:         s ← naDruhou[c, modul]
9:         m ← krat[s, a, modul]
10:        if rozvoj[[i]] = 0 then
11:            c ← s
12:        else
13:            c ← 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.

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

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

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

4.1.4 Možná zlepšení

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

4.1.4.1 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 18 *Nechť α je* generátor multiplikativní grupy *tělesa $\mathbb{F}$. Potom říkáme, že α je* primitivní prvek *tělesa $\mathbb{F}$.*

Důsledek Každý prvek tělesa $\mathbb{F}$ – kromě *nulového* prvku *aditivní grupy* – lze vyjádřit jako α^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 α^{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 jiné operaci s mocninami *primitivního prvku*.

Pro použití tohoto zrychlení výpočtů je tak nutné připravit v paměti programu překladové *log-* a *antilogaritmické* 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.
  - Tato tabulka 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*.

4.1.4.2 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.

4.2 Ireducibilní binární Goppa kódy

Pro implementaci kryptosystému jsme zvolili ireducibilní Goppa kódy, které jsme popsali v kapitole 2.3. V podkapitole 4.2.1 popíšeme algoritmy, které slouží pro vygenerování kódu dle zadaných parametrů a v podkapitole 4.2.2 *Pattersonův* algoritmus pro dekódování a (opravu) chyb vzniklých při přenosu.

4.2.1 Generování Goppa kódu

Parametry *ireducibilního binárního Goppa kódu* (n, k, t) jsou jednoznačně určené 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 4.1 a pro vygenerování matic a podpory jsou implementovány dílčí funkce popsané níže.

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 a podporu L. Výpočet je pak proveden pomocí *aplikování* (`Map`) funkcí dosazení do polynomu a inverze v tělese na prvky seznamu L.[29]

Matice V
Vandermondova $n \times t$ 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

[29] Zde by šlo dosazení do polynomu všech prvků z L zrychlit stejným způsobem, jako byl uvedený u *Chienova* hledání kořenů v kapitole 2.3.2.

Algoritmus 8 Generování Goppa kódu

```
1: function generujGoppaKod[ m, t ]
2:     n ← 2^m;   r ← tm;   k ← n − r
       // Ireducibilní polynom stupně m pro vnitřní modul
3:     modul_vnitrni ← {ireducibilniPolynom[2, m + 1], 2}
       // Goppův polynom g stupně t
4:     g ← ireducibilniPolynom[modul_vnitrni, t + 1]
5:     modul ← {g, modul_vnitrni}

6:     L ← generujPodporuL[m]                    ▷ Generování posloupnosti L
7:     V ← maticeV[podporaL, t, modul_vnitrni]   ▷ Vandermondova matice
8:     D ← maticeD[podporaL, modul]              ▷ Diagonální matice

9:     H ← dotNadF[V, D, modul_vnitrni]          ▷ Násobení matic nad GF(2^m)
10:    H ← Flatten[Transpose /@ H, 1]            ▷ Rozbalení prvků matice H
       // Převod matice H na G – ortogonální doplněk
11:    G ← NullSpace[H, Modulus → 2]

12:    X ← {jednotkovyPolynom[...], nulovyPolynom[...]}   ▷ Polynom x
       // předpočítané dílčí syndromy (x − L_i)^−1
13:    for i ← 1 ... n do                        ▷ Ve skutečnosti pomocí funkce Map
14:        syndromyL[[i]] ← inverze[plus[X, L[[i]], 2], modul]
15:    end for

16:    return {G, modul, L, syndromyL}
17: end function
```

a každý další řádek vypočítat přinásobením příslušného L_i k řádku předešlému. Tento výpočet je realizován funkcí softwaru *Mathematica* `NestList`.

4.2.2 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*, jak je popsaný v 4.2.2.

Druhá mocnina

Druhá mocnina je implementována přímo v rámci algoritmu, jelikož je třeba vynechat redukci polynomu. Tato druhá mocnina se vypočítá[30] stejným způsobem, jako byl uveden v kapitole 4.1.3.5.

[30] Výpočet druhé mocniny pomocí $kratBezRedukce[a, a, modul_{vnitrni}]$ není efektivní.

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** $s = \mathbf{0}$ **then** ▷ Pokud je syndrom nulový, chyba nenastala
5: $e \leftarrow nulovyPolynom[2, n]$
6: **else**
7: $X \leftarrow \{jednotkovyPolynom[\ldots], nulovyPolynom[\ldots]\}$ ▷ Polynom x
// $r = \sqrt{s(x)^{-1} - x} \mod g(x)$
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** $i \in 1 \ldots n$ **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** $\{d, e\}$ ▷ Vrátí dekódovanou zprávu i chybový vektor
20: **end function**

Modifikovaný *EEA*

Rozložení polynomu r je realizováno modifikovaným *rozšířeným Euklidovým algoritmem*, jak bylo popsáno v kapitole 2.3.2.

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.[31]

4.2.3 Možná zlepšení

Nejnáročnější část dekódování je hledání kořenů *lokátoru chyb* σ. Opakované dosazování do polynomu lze efektivněji implementovat pomocí *Chienova* hledání kořenů, či případně faktorizací polynomu σ. Více informací ohledně tohoto problému jsme uvedli v kapitole 2.3.2.

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 sestrojit 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 kódu. 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.

Stejně tak je při každém dekódování počítán počet prvků tělesa pro vypočítání odmocniny. Tento počet prvků – respektive číslo, na které je nutné prvek umocnit, abychom nalezli 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.

[31] Jedná se vlastně o řešení soustavy k rovnic pro k neznámých určených vybranou maticí $G_{\mathcal{K}}$. Je jistě nejjednodušší vybrat si takové dimenze $\mathcal{K}$, že $G_{\mathcal{K}}$ je jednotková matice.

4.3 McEliece

4.4 Měření

Závěr

draft [37]

Literatura

[1] Robert J. McEliece, A Public-Key Cryptosystem Based on Algebraic Coding Theory v *JPL Deep Space Network Progress Report 42-44* Jenuary and February 1978, strany 114–116. Dostupné online `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`

[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%2F978-3-642-20901-7_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

[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

[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 Tillich. 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

[17] Jean-Charles Faugre, Ayoub Otmani , Ludovic Perret, Frederic de Portzamparc, Jean-Pierre Tillich. *Structural Cryptanalysis of McEliece Schemes with Compact Keys*. IACR Cryptology ePrint Archive, 2014. Dostupné online https://eprint.iacr.org/2014/210.pdf

[18] Valery D. Goppa. A New Class of Linear Correcting Codes v *Problemy Peredachi Informatsii*, vol. 6, strany 24-30. 1970.

[19] Stefan Heyse. *Code-based Cryptography*: Implementing the McEliece Scheme on Reconfigurable Hardware. Diplomová práce. Ruhr-University Bochum, 2009.

[20] 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`

[21] 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`

[22] Jaroslav Kotil. *Goppa kódy a jejich aplikace*. Diplomová práce. Matematicko-fyzikální fakulta Univerzity Karlovy, Praha, 2013.

[23] Miroslav Kratochvíl. *Implementation of cryptosystem based on error-correcting codes*. Bakalářská práce. Matematicko-fyzikální fakulta Univerzity Karlovy, Praha, 2013.

[24] 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`

[25] 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`

[26] Robert McEliece. *The Theory of Information and Coding*. Encyclopedia of Mathematics and its Applications, vol. 3. Addison-Wesley, 1977.

[27] 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 Survey 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/`

[28] 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

[29] Rafael Misoczki, Jean-Pierre Tillich, 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

[30] Harald Niederreiter. Knapsack-type cryptosystems and algebraic coding theory v *Problems of Control and Information Theory 15*, strany 19-34. 1986

[31] 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

[32] 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.

[33] 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

[34] 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

[35] 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

[36] 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

[37] 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/

[38] 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

[39] 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

[40] 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

[41] 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

[42] 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

[43] Jan Mareš. *Algebra* – Úvod do obecné algebry. Skripta. ČVUT, 1999.

[44] Jiří Pytlíček. Lineární algebra a geometrie. Skripta. ČVUT, 2008.

[45] Přednášky MI-AAK

[46] Přednášky MI-BHW

[47] Přednášky MI-KRY

[48] Wolfram Mathematica

Příloha **A**

Seznam použitých zkratek

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 šifrového 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 *Gallois 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

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

S&M Algoritmus *Square-and-Multiply*

Příloha B

Obsah přiloženého CD

- `readme.txt` ..... stručný popis obsahu CD
- `exe` ..... adresář se spustitelnou formou implementace
- `src`
  - `impl` ..... zdrojové kódy implementace
  - `thesis` ..... zdrojová forma práce ve formátu LaTeX
- `text` ..... text práce
  - `thesis.pdf` ..... text práce ve formátu PDF
  - `thesis.ps` ..... text práce ve formátu PS