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.

24. dubna 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 24. dubna 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, polynomy, Wolfram Mathematica

Obsah

Seznam obrázků

Seznam tabulek

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 *Niederreiterovo* schéma – „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 kapitole jsou probrány definice a algoritmy nutné pro práci s *konečnými tělesy* a *polynomy* nad konečným tělesem. V práci se předpokládá základních znalostí z oblasti *algebry*. Pro tato témata je doporučena literatura [28, 27, 24, 25, 21] (kde lze též najít většinu důkazů následujících vět).

Poznámka
Algoritmy zmíněné v následujících kapitolách jsou detailně – včetně pseudokódu – popsány v kapitole zabývající se implementací (kapitola 4).

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 bude ukázáno dále, je vhodné prvky 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* $GF(p)$ *(řádu maximálně* $n-1$*)*. Na prvek a je též možné se dívat 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 se mezi těmito reprezentacemi prvků nadále volně přechází, jak bude v daném kontextu potřeba[3].

1.3 Operace v tělese $GF(p^n)$

V následujících sekcích jsou probrány operace potřebné pro počítání s tělesy $GF(p^n)$. Konkrétní zvolené algoritmy a jejich implementace je detailně popsána v kapitole 4.

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 $GF(p)$:

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

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*.

[3] V některých materiálech se používá i obráceného zápisu $(a_0a_1\ldots a_{n-1})$.

Při násobení prvků se opět využije 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* (v tělese $GF(p)$.

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 určení tělesa $GF(p^n)$ určit *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:*

$$A(x) \cdot B(x) = P(x) \Rightarrow (deg(A(x)) = 0) \vee (deg(B(x)) = 0)$$

Neboli pro *ireducibilní* polynom platí, že neexistuje rozklad na polynomy nad $GF(p)$ stupně alespoň 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 *Pro prvek a tělesa T a číslo $n \in \mathbb{N}$ je operace umocňování definována 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 $GF(p^n)$.

1.3.4 Inverze

Inverzi v *grupě* lze obecně definovat následovně:

Definice 4 (Inverze) *Nechť a je prvkem a $\mathbb{O}$ neutrálním prvkem* grupy *$G = (M, \circ)$. Prvek $\bar{a}$ je* inverzí *prvku a, pokud platí následující rovnice:*

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

1.3.4.1 Aditivní inverze

Inverze v *aditivní grupě* je značena 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 $GF(p)$).

1.3.4.2 Multiplikativní inverze

Inverze v *multiplikativní grupě* je značena 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 např. *algoritmus Itoh-Teechai-Tsujii* (*ITT*) [25, 23].

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].

[4] Někdy uváděno jako dlouhé dělení.

1.4 Rozšířená tělesa

Prvotěleso

Kapitola 2

Lineární kódy

2.1 Kódování

2.2 Lineární kódy

2.2.1 Hammingovy kódy

2.3 Goppa kódy

Ireducibilní binární Goppa kódy
Pattersonův algoritmus
Chien search , ...
List Decoding [13]

Kapitola 3

Kryptosystém McEliece

Kryptosystém *McEliece* je asymetrický šifrovací algoritmus, publikovaný poprvé v roce *1978 Robertem McEliece* [1]. V následujících kapitolám jsou probrány algoritmy navržené *Robertem McEliece*, dále *Niederreiterovo schéma* – varianta pro získání elektronického podpisu – a nakonec jsou zmíněny slabiny a existující útoky na tento kryptosystém.

Poznámka
V této kapitole je nadále předpokládáno 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. Zvolí se *lineární kód*[5] (n, k), opravující t chyb (a pro který je znám efektivní dekódovací algoritmus) s odpovídající $k \times n$ *generující maticí* G.
2. Vygeneruje se *náhodná* $k \times k$ *regulární* matice S.
3. Vygeneruje se *náhodná* $n \times n$ *permutační* matice P.
4. Vypočítá se $k \times n$ matice $\hat{G} = SGP$.

[5] V článku je kryptosystém definovaný pro libovolný *lineární kód* opravující zvolený počet chyb a jsou zmíněny *Goppa* kódy jako vhodný příklad k použití. Jak bude ukázáno 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 a matice S a P jsou *soukromý klíč*.

3.1.2 Algoritmy pro šifrování a dešifrování

3.1.2.1 Šifrování

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

1. Vygeneruje se náhodný vektor z délky n s *Hammingovou vahou* maximálně t[6].
2. Šifrovaná zpráva c délky n se sestrojí následujícím způsobem:
$$c = m\hat{G} + z$$

3.1.2.2 Dešifrování

Obdržená zašifrovaná zpráva c (délky n) se dešifruje následujícím způsobem:

1. Vypočítá se vektor $\hat{c}$ délky n: $\hat{c} = cP^{-1}$.
2. Vektor $\hat{c}$ se dekóduje zvoleným kódem na vektor $\hat{m}$
$\hat{m} = Dek_G(\hat{c})$
3. Vypočítá se původní zpráva m: $m = \hat{m}S^{-1}$

3.1.2.3 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.

[6] V některých pozdějších pracích na toto téma je uvedeno právě t.

3.1.3 Základní vlastnosti kryptosystému

V této kapitole jsou probrány základní fakta a vlastnosti *kryptosystému*. Jsou zde popsány 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 $t = 50$[7]. 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 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 $\approx$ 26 kb. Celkem se jedná o přibližně 300 kb dat pro uložení soukromého klíče

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

Metody snížení velikosti klíčů *kryptosystému McEliece* jsou jedním z hlavních překážek pro rozšíření algoritmu a také jedním z hlavních cílů zkoumání tohoto *kryptosystému* a věnuje se jim kapitola 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 $\hat{m}$.

3.1.4 Bezpečnost kryptosystému

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

[7] Jak bude zmíněno dále, velikost těchto parametry je pro dnešní použití nedostatečná.

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, ale množství jednotlivých matic je pro velká n a k obrovské, a získat tak původní matice hrubou silou je *neschůdné*[8].

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 [4].

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.

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

3.2 Niederreiterovo schéma

V roce 1986 publikoval *Harald Niederreiter* v [2] 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. Zvolí se *lineární kód* (n, k), opravující t chyb s odpovídající $(n-k) \times n$ *kontrolní maticí* H.

2. Vygeneruje se *náhodná* $(n-k) \times (n-k)$ *regulární* matice S.

3. Vygeneruje se *náhodná* $n \times n$ *permutační* matice P.

4. Vypočítá se $(n-k) \times n$ matice $\hat{H} = SHP$.

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 *soukromý klíč*.

3.2.2 Algoritmy pro šifrování a dešifrování

3.2.2.1 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 $n-k$) se spočte 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*[9] 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*.

[9] Zde nejsou na mysli samoopravné kódy, ale pouze jednoznačné zakódování zprávy.

3.2.2.2 Dešifrování

Obdržená šifrová zpráva c se dešifruje následujícím způsobem:

1. Vypočte se vektor $\hat{c}$ délky $n-k$: $\hat{c} = c\left(S^T\right)^{-1}$
2. Pomocí dekódovacího algoritmu použitého kódu se z $\hat{c}$ získá chybový vektor $\hat{m}$ (délky n).
3. Původní zpráva m se získá 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 $\left(S^T\right)^{-1}$.

3.2.2.3 Důkaz dešifrování

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

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

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

- 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

Niederreiterovo schéma je variantou asymetrického kryptosystému založeného na lineárních kódech, jak je použito u kryptosystému *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 [3] 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 [2, 10].

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 [10] publikován postup pro získání elektronického podpisu za pomocí asymetrického kryptosystému založeného na samoopravných kódech.

3.3.1 Překážky pro použití McEliece pro podepisování

Aby bylo možné 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 [10].

3.3.2 Schéma pro elektronický podpis

V roce 2001 autoři *Courtois* a spol. v [10] 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 *Niederreiterovo* schéma.

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[10] a pro bezpečnost odpovídající 80 bitům symetrické šifry jsou zvoleny

[10] S ohledem na útok *Canteaut-Chabaud* [8].

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 bude ukázáno dále, hodnota i bude třeba pro ověření podpisu a podpis je tak dvojice (z, i).

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 $S = (z|i_0)$

Délka podpisu
Délka podpisu závisí na uložení dat z a i_0. Vektor z je chybový vektor odpovídajícího samoopravného kódu. Jeho *Hammingova váha* je tedy maximálně t a je tedy velmi řídký. Existuje pouze $\binom{n}{t}$ vektorů *váhy* t a délky n a je tedy možné tento řídký vektor komprimovat. V [10] 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 Algoritmus pro podepisování

Podpis je sestrojen následujícím způsobem:

- Sestrojíme *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.

3.3.4 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.

3.3.5 Poznámky

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.

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* [4]

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. Volba bezpečných parametrů je probrána v kapitole 3.4.2.

3.4.1 Útoky na McEliece

V této kapitole jsou uvedeny některé z útoků na kryptosystém *McEliece*. Dle [9] 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* [17]. 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[11].

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.

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

V původním článku, kde bylo definováno *Niederreiterovo* schéma, bylo navrženo použití *obecných Reed-Solomon* (*GRS*) kódů [2]. V [**?**] 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* [18, 19, 20].

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* [17].

Definice 5 *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 [13].

3.4.1.3 Útok s informační množinou

Útok s informační množinou (*Information Set Decoding attack – ISD*), který byl popsán již v původním článku *Roberta McEliece* [1] a zmíněn v kapitole 3.1.4. Tento útok byl formalizován a zobecněn v [16].

Útok je založen na výběru k sloupců (dimenzí) (množina K) 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 [16].

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

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

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 [6, 7, 8] nejdříve hledají kódová slova v redukovaném kódu $\mathcal{K}'_S$, který vznikne výběrem náhodnou množinou dimenzí S z matice $\hat{G}'$. Poté se tato kódová slova rozšíří do původního kódu $\mathcal{K}'$ a zkontrolují, zda mají požadovanou *váhu*.

Algoritmy představené autory *Leon* [6], *Stern* [7] a *Canteaut* a *Chabaud* [**?**] 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 [**?**] či útok založený na *bodových mřížích* [15]. Jako další zdroje pro zkoumání těchto útoků jsou doporučeny články [13, 9].

3.4.2 Bezpečné parametry

Tabulky z aktuálních článků

3.4.3 Slabiny kryptosystému

V této kapitole jsou shrnuty 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, která je popsaná 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*.

Zašifrovaná zpráva c_1 veřejným klíčem $\hat{G}$ byla zkonstruována (dle definice) $c_1 = m_1\hat{G} + z$, kde z je náhodný chybový vektor. Pokud je tato zpráva c_1 zachycena, je možné ji pozměnit následujícím způsobem:

- Připraví se (otevřená) zpráva m_1
- Tato zpráva se „zašifruje“ veřejným klíčem $\hat{G}$, ale nepoužije se chybový vektor z: $c_2 = m_2\hat{G}$
- K původní zašifrované zprávě c_1 se přičte nová zpráva c_2: $c = c_1 + c_2$
- Odešle se vzniklá zpráva 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* [26]. 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 [5]. Pro každé šifrování je generován náhodný (a pravděpodobně tedy jiný) chybový vektor z. Sečtením dvou různých šifrových textů jedné zprávy se tak získá 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 $\binom{n}{t}$[12] – 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.

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):

[12] Por praktické parametry kryptosystému platí $n \gg t$.

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

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[13].

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.

3.5 Moderní varianty a úpravy

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

[13] Podobně jako např. útok *Paddding Oracle* u blokových šifer [26].

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 jsou zmíněny v kapitole 3.5.1.2.

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

3.5.1.1 Význam matic S a P

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* [11, 12, 14].

Definice 6 *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 7 *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 [11] 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* [11].

3.5.2 CCA2-odolné konverze

3.5.3 Odolnost vůči kvantovým počítačům

Kapitola 4

Implementace

Pro implementaci kryptosystému *McEliece* v této práci byl zvolen software *Wolfram Mathematica* [29]. Tento software byl zvolen 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.

Celková práce byla 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

Tato kapitola pojednává o implementaci *binárních konečných těles* včetně jejich *rozšíření*. Jsou zmíněna existující řešení v softwaru *Mathematica*, zvolená implementace a popis implementovaných algoritmů.

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 jsou popsány 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*[14] *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*[15]. 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*[16] 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:

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

- Prvky *konečných těles* jsou reprezentovány *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 1 Sčítání polynomů

```
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 [22]. 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 2 Redukce polynomu 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 $x^{12} + x^8 + x^7 + x^5 + x^4 + x^3 + 1$ polynomem $x^4 + x + 1$ (nad tělesem $GF(2)$):

$$
\begin{array}{l}
1000110111001 \mod 10011: \\
\hline
1\,0\,0\,0\,1\,1\,0\,1\,1\,1\,0\,0\,1 \\
1\,0\,0\,1\,1\,0\,0\,0\,0\,0\,0\,0\,0 \\
0\,0\,0\,1\,0\,0\,1\,1\,0\,0\,0\,0\,0 \\
0\,0\,0\,0\,0\,1\,0\,0\,1\,1\,0\,0\,0 \\
0\,0\,0\,0\,0\,0\,1\,0\,0\,1\,1\,0\,0 \\
\hline
1\,1\,0\,1
\end{array}
$$

$\Rightarrow |1000110111001|_{10011} = 1101$

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 3 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 $x^3 + x + 1$ polynomem $x^4 + x^2 + 1x + 1$ (nad tělesem $GF(2)$):

$$
\begin{array}{l}
1011 \cdot 10111: \\
1(x^4)\,1\,0\,1\,1\,0\,0\,0\,0 \\
0(x^3)\,0\,0\,0\,0\,0\,0\,0\,0 \\
1(x^2)\,0\,0\,1\,0\,1\,1\,0\,0 \\
1(x^1)\,0\,0\,0\,1\,0\,1\,1\,0 \\
\underline{1(x^0)\,0\,0\,0\,0\,1\,0\,1\,1} \\
\qquad\quad 1\,0\,0\,0\,0\,0\,0\,1
\end{array}
$$

⇒ Výsledek operace násobení modulo polynom g se získá redukcí polynomu 10000001 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[17].

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

Příklad *Rozšířený Euklidův algoritmus* pro výpočet *inverze* polynomu $x^3 + x^2 + 1$ *modulo* $x^6 + x + 1$ (nad tělesem $GF(2)$):

Podíl	Zbytek	Koeficienty	
	1000011	0	1
	1101	1	0
1110	101	−1110	1
11	10	10011	−11
10	1	−101000	111

$\Rightarrow |1101^{-1}|_{1000011} = 101000$

Poznámka
Poslední sloupec tabulky se v algoritmu nepočítá, je zde uveden pouze pro úplnost.

[17] Zde je vidět, že pro výpočet inverze v tělese $GF(q^n)$ je třeba vypočítat inverzi v tělese $GF(q)$.

Algoritmus 4 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 1 *Nechť* $A = (a_n \dots a_2 a_1 a_0)$ *je prvek tělesa s* charakteristikou 2*, potom platí:*

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

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 5 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
```

Důkaz

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

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[18]. Algoritmus má vstupy polynom a a exponent e. Exponent se vyjádří jako číslo v *binární* soustavě a poté algoritmus provádí cyklus přes bity tohoto rozvoje. V každém kroku se mezivýsledek umocní na druhou a v případě, že je odpovídající bit exponentu 1, přinásobí se původní číslo a.

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.

[18] Uváděna jako *MSB* – z anglického *most significant bit*

Algoritmus 6 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
```

Příklad *Square-and-Multiply* pro výpočet $(x^3+1)^{26}$ *modulo* x^6+x+1 (nad tělesem $GF(2)$):

Op.	Mocnina dek.	Mocnina bin.	Výpočet	Výsledek
	1	1		1001
S	2	1	1001^2	10
M	3	11	$10 \cdot 1001$	10010
S	6	110	10010^2	1000
S	12	1100	1000^2	11
M	13	1101	$11 \cdot 1001$	11011
S	26	11010	11011^2	1010

$\Rightarrow |1001^{26}|_{1000011} = 1010$

4.1.4 Možná zlepšení

V této kapitole jsou nastíněny 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 8 *Nechť α je* generátor multiplikativní grupy *tělesa F. Potom říkáme, že α je* primitivní prvek *tělesa F.*

Důsledek Každý prvek tělesa 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 lze převést na součet mocnin *primitivního prvku*:

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

Podobným způsobem je možné 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 se získá prvek α^{n_c}, kde n_c je celé číslo v rozsahu od 0 do $N - 1$.

Reprezentací prvků pomocí odpovídajících mocnin *primitivního prvku* je tak možné vyhnout se 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*, je možné 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.

Tohoto vylepšení se dá 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* se převádí 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. 4) 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

4.3 McEliece

4.4 Měření

Závěr

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

[3] 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`

[4] 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.

[5] 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.

[6] 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`

[7] 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`

[8] 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

[9] 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

[10] 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

[11] 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

[12] Olga Paustjan. *Post Quantum Cryptography on Embedded Devices*: An Ecient Implementation of the McEliece Public Key Scheme based on Quasi-Dyadic Goppa Codes. Ruhr-University Bochum, 2010.

[13] 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

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

[15] 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

[16] 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

[17] 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`

[18] 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`

[19] 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`

[20] 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`

[21] 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`

[22] 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, Numbers 1-3, strany 3-32. Ruhr-Universitat Bochum, 2006. Dostupné online: `http://www.emsec.rub.de/research/publications/efficient-software-implementation-finite-fields-ap/`

[23] 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`

[24] Přednášky BI-LIN

[25] Přednášky MI-BHW

[26] Přednášky MI-KRY

[27] Přednášky MI-MKY

[28] Přednášky MI-MPI

[29] Wolfram Mathematica

Příloha **A**

Seznam použitých zkratek

CCA2 *Adaptive Chosen Ciphertext Attack* – útok s adaptivní volbou šifrového textu

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

MSB *Most Significant Bit/Byte* – nejvíce významný bit/bajt

OAEP *Optimal asymmetric encryption padding* – schéma pro asymetrické šifrování

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