SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: 92318

Implementácia QC-MDPC McEliece kryptosystému nad GF(p)

Diplomová práca

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: 92318

Implementácia QC-MDPC McEliece kryptosystému nad GF(p)

Diplomová práca

|  |  |
| --- | --- |
| Študijný program : | Aplikovaná informatika |
| Číslo študijného odboru: | 2511 |
| Názov študijného odboru: | 9.2.9 Aplikovaná informatika |
| Školiace pracovisko: | Ústav informatiky a matematiky |
| Vedúci záverečnej práce: | Mgr. Tomáš Fabšič, PhD. |
|  |  |

ZADANIE DIPLOMOVEJ PRÁCE

Študent Bc. Juraj Paška

ID študenta: 92318

Študijný program: Aplikovaná informatika

Študijný odbor: Informatika

Vedúci práce: Mgr. Tomáš Fabšič, PhD.

Miesto vypracovania: Ústav informatiky a matematiky

Názov práce: Dekódovanie QC-MDPC kódov s nízkou pravdepodobnosťou

chyby

Špecifikácia zadania: Pokrok vo vývoji kvantového počítača má vážne dôsledky aj pre kryptografiu. Je známe, že dostatočne vykonné kvantové počítače budú vedieť efektívne riešiť problém faktorizácie čísla na prvočísla a problém diskrétneho logaritmu. Na náročnosti riešenia týchto problémov je založená bezpečnosť v súčasnosti používaných asymetrických kryptosystémov (napríklad RSA). To znamená, že v prípade existencie dostatočne výkonného kvantového počítača by súčasné asymetrické kryptosystémy už neboli bezpečné. Niektoré odhady hovoria, že takto vykonné kvantové počítače by mohli existovať už o 10 rokov. Je preto dôležité, pracovať na vývoji nových asymetrických kryptosystémov, ktoré budú odolné voči útokom kvantového počítača, a ktoré by mohli nahradiť súčasné asymetrické kryptosystémy. Na dôležitosť tejto témy upozornil aj americký inštitút pre štandardy a technológiu NIST v správe Report on Post-Quantum Cryptography (správa je dostupná na https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf ). NIST zároveň vyhlásil súťaž Post-Quantum Cryptography Standardization Process ( https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Post-Quantum-Cryptography-Standardization ) s cieľom navrhnúť nové kryptografické štandardy odolné voči kvantovým počítačom. Do tretieho kola súťaže sa dostal aj kryptosystém Bike, ktorý je založený na QC-MDPC McEliece kryptosystéme. V pôvodnom QC-MDPC McEliece kryptosystéme sa výpočty vykonávajú nad GF(2). V roku 2016 Guo a Johansson navrhli verziu QC-MDPC McEliece kryptosystému fungujúcu nad GF(p). Cieľom tejto práce je implementovať verziu QC-MDPC McEliece kryptosystému od autorov Guo a Johansson v jazyku C alebo C++.

Úlohy:

1. Oboznámte sa s princípmi fungovania QC-MDPC McEliece kryptosystému nad GF(p).

2. Implementujte QC-MDPC McEliece kryptosystém nad GF(p) v jazyku C alebo C++ a otestujte vašu implementáciu.

Literatúra:

1. Guo, Q. and Johansson, T., 2016, July. A p-ary MDPC scheme. In 2016 IEEE International Symposium on Information Theory (ISIT) (pp. 1356-1360). IEEE.

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE

FAKULTA ELEKTROTECHNIKY A INFORMATIKY

|  |  |
| --- | --- |
| Študijný program : | Aplikovaná informatika |
| Diplomová práca: | Implementácia QC-MDPC McEliece kryptosystému nad GF(p) |
| Autor: | Bc. Juraj Paška |
| Vedúci záverečnej práce: | Mgr. Tomáš Fabšič, PhD. |
| Miesto a rok predloženia práce: | Bratislava 2023 |

Táto záverečná práca sa zaoberá analýzou a implementáciou asymetrického kryptosystému založeného na teórii kódovania. Je totiž známe, že systémy založené na teórii kódovania sú vo všeobecnosti odolnejšie voči potenciálnym útokom kvantových počítačov, než systémy založené na faktorizácii veľkých čísel, alebo na probléme diskretného logaritmu. Konkrétne ide o modifikáciu QC-MDPC systému s použitím nebinárnej abecedy od autorov Guo a Johanson. Dôvodom prečo sa autori zaoberajú použitím nebinárnej abecedy je najmä redukcia veľkosti verejného kľúču a taktiež rozsiahlejšie možnosti v oblasti dešifrovania v porovnani s klasickým McEliece kryptosystémom s použitím Goppa kódov. Takýto systém bol ako súčasť diplomovej práce implementovaný s použitím C++ knižnice NTL a C knižnice FLINT.

Kľúčové slová: post-kvantová kryptografia, QC-MDPC, asymetrická kryptografia, McEliecov kryptosystém

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

|  |  |
| --- | --- |
| Study Programme: | Applied Informatics |
| Bachelor Thesis: | Implementation of QC-MDPC McEliece cryptosystem over GF(p) |
| Autor: | Bc. Juraj Paška |
| Supervisor: | Mgr. Tomáš Fabšič, PhD. |
| Place and year of submission: | Bratislava 2022 |

This diploma thesis concerns about analysis and implementation of asymetric cryptosystem based od Coding Theory. Such cryptosystem is in general more resistant against potential attack of quantum computers than the current systems based on factorization of large numbers or on problem of discrete logarithm. This particular system is modification of QC-MDPC cryptosystem with non-binary alphabet. Main reason why authors - Guo, Johanson, decided to analyse such a cryptosystem, is reduction of public key size and the posibility of usage of new aproach to decoding. Implemetation of this P-ary cryptosystem was done as part of this thesis, with usage of C++ library NTL and C library FLINT.

Key words: postquantum cryptography, asymetric cryptography, MCEliece cryptosystem, QC-MDPC

Vyhlásenie autora

Podpísaný Bc. Juraj Paška čestne vyhlasujem, že som Diplomovú prácu Implementácia QC-MDPC McEliece kryptosystému nad GF(p) vypracoval na základe poznatkov získaných počas štúdia a informácií z dostupnej literatúry uvedenej v práci.

Uvedenú prácu som vypracoval pod vedením Mgr. Tomáš Fabšič, PhD.

V Bratislave dňa 04.04.2023

..................................................

podpis autora

Poďakovanie

Touto cestou by som sa chcel poďakovať vedúcemu diplomovej práce Mgr. Tomášovi Fabšičovi, PhD., za odborné vedenie tejto záverečnej práce, bez ktorého by nebolo možné túto prácu zhotoviť.

# Obsah

[Zoznam obrázkov a tabuliek 4](#_Toc128812278)

[Zoznam príkladov a algoritmov 5](#_Toc128812279)

[Zoznam skratiek a značiek 6](#_Toc128812280)

[1. Úvod a motivácia 12](#_Toc128812281)

[2. Analýza 13](#_Toc128812282)

[2.1 Teória kódovania 13](#_Toc128812283)

[2.1.1 Základné pojmy 13](#_Toc128812284)

[2.1.2 Cyklické kódy 16](#_Toc128812285)

[2.1.3 Kvázi-cyklické kódy 17](#_Toc128812286)

[2.2 LDPC a MDPC kódy 17](#_Toc128812287)

[2.3 QC-MDPC kódy 18](#_Toc128812288)

[3. QC-MDPC nad GF(P) 20](#_Toc128812289)

[3.1 Popis QC-MDPC kódov nad GF(P) 20](#_Toc128812290)

[3.1.1 Konštrukcia MDPC kódov nad GF(P) [6] 20](#_Toc128812291)

[3.1.2 Konštrukcia QC-MDPC kódov nad GF(P) [6] 21](#_Toc128812292)

[3.2 Schéma QC-MDPC kódov nad GF(P) [7] 21](#_Toc128812293)

[4. Dekódovanie P-ary QC-MDPC kódov 24](#_Toc128812294)

[4.1 Algoritmus 1 [7] 24](#_Toc128812295)

[4.2 Heuristický dekóder – TODO 33](#_Toc128812296)

[5. Návrh riešenia a testovanie 34](#_Toc128812297)

[5.1 Dekódovacia chyba Algoritmu 1 34](#_Toc128812298)

[Záver 38](#_Toc128812299)

[Zoznam použitej literatúry 39](#_Toc128812300)

[Prílohy 40](#_Toc128812301)

# Zoznam **obrázkov, tabuliek a definícii**

[Obrázok 1: Intervaly pre výpočet pozícii chybového vektoru ............................................28](#_Obrázok_1:_Intervaly)

[Obrázok 2: Intervaly pre významnú pozíciu h0(1) ............................................................... 29](#Obrázok2)

[Obrázok 3: Intervaly pre významnú pozíciu h1(1) ............................................................... 29](#Obrázok3)

[Obrázok 4: Parametrizovanie algoritmu DecideThr v iteráciách ........................................32](#_Obrázok_4:_Parametrizovanie)

[Tabuľka 1: Súčiny členov e­­Ti a hi, kde i je rôzne od 0 a rôzne od k ...................................27](#_Tabuľka_1:_Súčiny)

[Tabuľka 2: Odhad hodnôt pozícii syndrómu s.....................................................................28](#_Tabuľka_2:_Odhad)

[Tabuľka 3: Dekódovacia chyba Algoritmu 1, pri k = 491, q = 345 ....................................36](#_Tabuľka_3:_Dekódovacia)

[Tabuľka 4: Dekódovacia chyba Algoritmu 1, pri k = 491, q = 360 ....................................37](#_Tabuľka_4:_Dekódovacia)

[Definícia 1: Lineárny kód ....................................................................................................13](#_Definícia_1:_Lineárny)

[Definícia 2: Generujúca matica ...........................................................................................14](#_Definícia_2:_Generujúca)

[Definícia 3: Kontrolná matica .............................................................................................14](#_Definícia_3:_Kontrolná)

[Definícia 4: Cyklický kód ...................................................................................................16](#_Definícia_4:_Cyklický)

[Definícia 5: Kvázi-cyklický kód .........................................................................................17](#_Definícia_5:_Kvázi-cyklický)

[Definícia 6: LDPC a MDPC Kódy .. ...................................................................................17](#_Definícia_6:_)

[Definícia 7: Syndróm správy ..............................................................................................19](#_Definícia_7:_)

# Zoznam algoritmov a príkladov

[Algoritmus 1 ........................................................................................................................24](#_Algoritmus_1_[7])

[Algoritmus Decide ..............................................................................................................30](#_Algoritmus_Decide_[7])

[Algoritmus DecideThr .........................................................................................................31](#_Algoritmus_DecideThr_[7])

[Príklad 1: Kódovanie lineárnym kódom s nebinárnou abecedou ........................................15](#_Príklad_1:_Kódovanie)

[Príklad 2: Zostrojenie generujúcej matice cyklického kódu ...............................................16](#_Príklad_2:_Zostrojenie)

[Príklad 3: Zostrojenie prvého riadku matice H ...................................................................25](#_Príklad_3:_Zostrojenie)

[Príklad 4: Výpočet syndrómu správy ..................................................................................27](#_Príklad_4:_Výpočet)

[Príklad 5: Úprava intervalu pre odhad hodnoty ei+k ..........................................................30](#_Príklad_5:_Úprava)

# **Zoznam skratiek a značiek**

OT – otvorený text

ZT – zašifrovaný text

e – chybový vektor

C – lineárny kód

H – kontrolná matica lineárneho kódu

G – generujúca matica lineárneho kódu

c – kódové slovo lineárneho kódu

u – riadkový vektor

n – počet stĺpcov generujúcej a kontrolnej matice, dĺžka kódového slova

k – počet riadkov generujúcej a kontrolnej matice, dĺžka otvoreného textu

r – počet redundantných prvkov kódového slova

DFR – Decoding Failure Rate - miera chyby dekódovania

n0 – počet cyklických blokov matice

s – syndróm správy

GF(2) – binárne Galoisové pole, označované tiež ako F2

GF(P) – Galoisové pole s P prvkami, označované tiež ako F­­p

Fpn – vektor dĺžky n, ktorý obsahuje prvky z GF(P)

QC-MDPC – kvázi cyklický kód so stredne hustou kontrolnou maticou

# Úvod a motivácia

Táto diplomová práca sa zoberá analýzou a implementáciou QC-MDPC McEliece kryptosystému od autorov Guo a Johanson [6]. V pôvodnom McEliece kryptosystéme sa výpočty vykonávajú nad GF(2). Spomínaní autori sa zaoberajú zovšeobecnenou verziou McEliece kryptosystému, v ktorom sa výpočty vykonávajú nad GF(P).

Súčasťou práce je aj implementačná časť napísaná v jazyku C++ a taktiež testovací projekt, ktorý implementovanú knižnicu používa.

Hlavnou motiváciou prečo dnes rôzne vedecké tímy skúmajú a implementujú nové asymetrické kryptosystémy je pokrok vo vývoji kvantových počítačov. Dnes používané systémy s verejným kľúčom sú často založené na probléme diskrétneho logaritmu (napr. Diffie-Hellman), alebo na faktorizácii veľkých čísel na súčin prvočiniteľov (napr. RSA). Je všeobecne známe, že dostatočne výkonný kvantový počítač dokáže tieto problémy efektívne riešiť s použitím Shorovho algoritmu [2].

# Analýza

McEliecov kryptosystém [1] je systém založený na teórii kódovania. Existuje viacero verzii tohto systému a to napríklad s použitím Goppa kódov, kódov s riedkou parity-check maticou – LDPC, so stredne hustou parity-check maticou - MDPC, alebo verzie s kvázi cyklickou parity-check maticou – QC-LDPC a QC-MDPC [8].

V tejto práci sa budeme zaoberať len systémom založeným na QC-MDPC kódoch, ktoré boli použité napríklad v návrhu BIKE [3]. Tento systém je jedným z kandidátov v súťaži NIST [4], ktorá má za úlohu vybrať vhodný asymetrický kryptosystém, odolný voči potenciálnym kvantovým útokom v budúcnosti.

V čase písania tejto práce už boli známe výsledky tretieho kola súťaže NIST, ktorého výsledkom boli prvé kryptosystémy pre štandardizáciu [5]. Do tohto výberu sa kryptosystém BIKE nedostal, avšak spolu s klasickým McEliecovým kryptosystémom sa dostal do ďalšieho, v poradí už štvrtého kola súťaže, kde bude vybratý algoritmus pre neskoršiu štandardizáciu. Spolu teda zo štyroch alternatívnych algoritmov, sú až 3 založené na teórii kódovania – tretím algoritmom je HQC.

Všetky výpočty sa v návrhu BIKE vykonávali nad GF(2). Autori Guo a Johanson v článku [6] deklarujú možnosť použitia menšieho verejného kľúča a taktiež použitie nových typov iteratívneho dekódovania, ak sa operácie vykonávajú nad GF(P).

## Teória kódovania

Pre popis takéhoto kryptosystému je najskôr potrebné uviesť základné koncepty teórie kódovania a konečných polí. V práci budeme označovať GF(P) ako konečné pole s prvočíselným počtom prvkom p.

### Základné pojmy

#### Definícia 1: Lineárny kód

*[n, n-r] Lineárny kód C nad poľom GF(P) je linéarny podpriestor vektorového priestoru Fpn s dimenziou* ***(n – r),*** *kde* ***n*** *je dĺžka kódového slova a****r*** *označuje redundanciu kódu.*

Takýto lineárny kód je popísaný generujúcou maticou G a kontrolnou maticou H. Matica G slúži na vytvorenie kódových slov dĺžky *n* a to ľubovoľnou lineárnou kombináciou riadkov tejto matice. Matica H slúži na overenie, či konkrétne kódové slovo patri ku kódu, ktorý je popísaný touto kontrolnou maticou. Platí, že po vynásobení kódového slova prislúchajúcou transponovanou kontrolnou maticou dostaneme nulový vektor – nulový syndróm správy.

#### Definícia 2: Generujúca matica

*Generujúca matica G lineárneho kódu C [n, n-r] je taká matica, ktorej ľubovoľná lineárna kombinácia jej riadkov tvorí kódové slovo patriace do kódu C. Matica má rozmer* ***(n - r)*** *x****n*** *a teda tvorí bázu lineárneho kódu C.*

Pre generujúcu maticu platí, že po vynásobení ľubovoľného riadkového vektora *u*maticou *G*, dostaneme kódové slovo *c.* Medzi riadkovým vektorom *u*a kódovým slovom *c* existuje bijekcia. Platí:

uG = c,

kde *u* je riadkový vektor dĺžky (n – r)a *c* je kódové slovo dĺžky n.

#### Definícia 3: Kontrolná matica

*Kontrolná matica H lineárneho kódu C [n, n-r] je taká matica, ktorej jadrom (kernelom) je lineárny kód C a táto matica je zároveň generujúcou maticou duálneho kódu C⊥ ku kódu C. Matica má rozmer* ***r x n****.*

V QC-MDPC McEliece krytposystéme sa generujúca matica G používa ako verejný kľúč a uvádza sa v systematickom redukovanom tvare, zatiaľ čo kontrolná matica H predstavuje súkromný kľúč a v dešifrovacích algoritmoch slúži na odstránenie chýb zo zašifrovanej správy. Platí vzťah:

GHT = 0,

kde 0 je matica núl s rozmerom k x (n – k). Z vyššie uvedených definícii a vzťahov vyplýva, že kódové slovo *c* patrí do lineárneho kódu C práve vtedy, keď platí:

cHT = 0,

kde 0 je vektor dĺžky (n – r) a predstavuje nulový syndróm správy *s.* Parameter k označuje počet riadkov generujúcej matice a platí vzťah

k = n – r,

kde *n* je počet stĺpcov generujúcej matice – a teda dĺžka kódových slov lineárneho kódu popísaného touto maticou – a *r* je počet reduntantých pozícii kódového slova. Redundantné pozície kódového slova vstupujú do procesu dešifrovania správy na kódové slovo lineárneho kódu, kde sa tieto reduntantné pozície snažíme vynulovať.

Pre lineárny kód C a jeho generujúcu maticu G a H platí, že matice G a H nie sú jedinečné, avšak k lineárnemu kódu C existuje jedinečný kód C⊥, ktorý je duálnym kódom ku kódu C a jeho generujúcou maticou je jedna z parity-check matíc kódu C.

#### Príklad 1: Kódovanie lineárnym kódom s nebinárnou abecedou

Uvažujme lineárny kód C nad poľom GF(5). Chceme zakódovať slovo m dĺžky (n – r), generujúcou maticou G, ktorá m = (n – r) riadkov a n stĺpcov. Dostávame kódové slovo c, ktoré je riadkovým vektorom dĺžky n. Generujúca matica je v systematickom redukovanom tvare a preto na prvých (n – r) pozíciách kódového slova c sú rovnaké členy ako pri slove m, ktoré sme kódovali. Posledných r pozícii kódového slova predstavuje redudantnú informáciu, ktorá nesie informáciu o tom, či kódové slovo patrí do lineárneho kódu.

[1 4 3 4] x [1 0 0 0 0 4 0 4] = [1 4 3 4 2 1 2 1]

[0 1 0 0 4 0 4 0]

[0 0 1 0 0 4 0 4]

[0 0 0 1 4 0 4 0]

Výsledné kódové slovo teda vzniklo ako lineárna kombinácia riadkov matice G a to:

c = m1 \* G1 + m2 \* G2 + m3 \* G3 + m4 \* G4­.

Vidíme, že pôvodnu správu môžme z kódového slova ľahko prečítať, preto pri šifrovaní nestačí správu zakódovať, ale je potrebné výsledné kódové slovo pozmeniť pridaním chybového vektora.

### Cyklické kódy

Pre popísanie QC-MDPC kryptosystému potrebujeme poznať štruktúru cyklických kódov. Síce sa v tomto systéme priamo nepoužívajú, ale cyklické bloky sú základným stavebným blokom kvázi-cyklických kódov popísaných v kapitole 2.1.3. Definície cyklických kódov preberáme z [9].

#### Definícia 4: Cyklický kód [9]

*Lineárny [n, n-r]–kód C sa nazýva cyklickým, ak s každým kódovým*

*slovom v0v1 . . . vn−1 ∈ C je kódové aj slovo vn−1v0v1 . . . vn−2. Týmto je definovaný cyklický posun znakov v slovách.*

Z definície vyplýva, že každé slovo ktoré vznikne posunutím akéhokoľvek kódového slova smerom doprava, je opäť kódovým slovom daného cyklického lineárneho kódu. Cyklické kódy sú taktiež popísané generujúcou maticou G a kontrolnou maticou H. Obe matice sa dajú zapísať v skrátenej forme a to pomocou prvého riadku. Ostatné riadky matíc vzniknú cyklickým posunom prvého riadku doprava.

#### **Príklad 2: Zostrojenie generujúcej matice cyklického kódu**:

Uvažujme lineárny kód C nad GF(5), ktorého generujúca matica G má 4 riadky a 8 stĺpcov. Teda:

n = 8,

k = 4,

r = n – k = 4.

Prvý riadok generujúcej matice môže vyzerať takto:

G­1=[4 0 0 0 1 0 0 0]

Potom celú generujúcu maticu G môžme zostrojiť k – 1 posunmi prvého riadku. Generujúca matica G vyzerá nasledovne:

G =[4 0 0 0 1 0 0 0]

[0 4 0 0 0 1 0 0]

[0 0 4 0 0 0 1 0]

[0 0 0 4 0 0 0 1]

### Kvázi-cyklické kódy

V QC-MDPC systéme sa používajú práve kvázi-cyklické kódy. Pri takýchto kódoch je kódovým slovom každé také slovo, ktoré vznikne cyklickým posunom ľubovoľného kódového slova doprava o *n0* pozícii. Parameter *n0* nám vyjadruje počet cyklických blokov, ktoré matice G a H kvázi-cyklického kódu horizontálne obsahujú.

#### Definícia 5: Kvázi-cyklický kód [6]

*[n, k] – Lineárny kód, kde n = n0 \* k, je kvázi-cyklický ak každý cyklický posun kódového slova o n0 pozícii je opäť kódové slovo. Generujúca a Kontrolná matica takéhoto lineárneho kódu sa skladá z cyklických blokov rozmeru* ***(k x k)****.*

Ako vyplýva z definície kvázi-cyklického kódu, na popísanie takéhoto kódu nám stačí uchovať prvý riadok Kontrolnej a Generujúcej matice takéhoto kódu, čo v praxi prináša značnú pamäťovú úsporu. Izomorfizmus medzi algebrou takýchto **(k x k)** cyklických matíc a polynómov v *Fp[x] / (xk - 1)* umožňujú realizovať maticové a vektorové výpočty ako operácie v popísanom okruhu polynómov.

## LDPC a MDPC kódy

Pôvodný McEliecov kryptosystém bol navrhnutý s použitím Goppa kódov[1]. Takýto kryptosystém sa vyznačuje použitím veľkého verejného kľúča, čo môže predstavovať problém pre zariadenia s obmedzenými výpočtovými prostriedkami. Kryptosystémy použitím riedkej kontrolnej matice – LDPC a stredne hustej kontrolnej matice – MDPC tieto problémy čiastočne riešili, no McEliecov kryptosystém s použitím QC-MDPC kódov zredukoval veľkosť kľúču ešte viac, pretože na výpočty v takomto systéme nám stačí poznať len prvé riadky matíc.

#### Definícia 6: LDPC a MDPC Kódy [6]

*LDPC kód (Low Density Parity Check)– je linárny kód s kontrolnou maticou s nízkou hustotou, zatiaľ čo MDPC kód (Moderate Density Partity Check) je lineárny kód s hustejšou, no stále riedkou kontrolnou maticou.*

V binárnych LDPC kódoch sa uvádza hustota kontrolnej matice H na úrovni 0,1% až 0,2%, pričom v hustejších binárnych MDPC kódoch sa táto hustota pohybuje na úrovni 0,5% a viac. **[10]**

## QC-MDPC kódy

V článku [8] je prezentovaný McEliecov kryptosystém s použitím QC-MDPC kódov. Hustejšia generujúca matica v tomto prípade umožnila vynechanie maskovacích prvkov tohto systému, čoho výsledkom bol jednoduchší systém s menej bezpečnostnými rizikami, ktoré práve tieto maskovacie techniky do systému prinášali.

Pôvodný QC-MDPC systém bol navrhnutý s použitím binárnej abecedy. Príklad takéhoto systému je návrh BIKE, ktorý je súčasťou štvrtého kola súťaže inštitútu NIST pre výber asymetrického kryptosystému odolného voči útokom kvantového počítaču. Pre lepšie pochopenie QC-MDPC kódov nad abecedou s P prvkami si môžeme popísať schému pôvodného systému.

#### Schéma QC-MDPC s binárnou abecedou

*Lineárny kód s binárnou abecedou, ktorý je popísaný kontrolnou maticou*

*H ∈ F2 (n - k) x n s w jednotkami v každom riadku matice, nazývame (n, k, w)-QC-MDPC binárny kód. Ku kódu a danej kontrolnej matici H prislúcha generujúca matica G ∈ F2 k x n. Obe matice majú n = n0 \* m stĺpcov, kde n0 je počet cyklických blokov v horizontálnom smere a m je rozmer tohto štvorcového cyklického bloku. Počet riadkov v matici G nám udáva parameter k a v pôvodnom systéme ho môžeme vyjadriť ako k = n - m. Počet riadkov matice H je rovný redundancii lineárneho kódu a môžeme ho vyjadriť ako*

*r = n – k.*

V takto popísanom systéme sa matica H používa ako súkromný kľúč pre potreby dešifrovania, zatiaľ čo matica G je zverejnená v systematickom redukovanom tvare ako verejný kľúč. Šifrovanie správy s dĺžkou *k* teda prebieha vynásobením tejto správy maticou G – čím vznikne kódové slovo lineárneho kódu s dĺžkou *n* – a pridaním určitého počtu chýb k tomuto slovu. Pridávanie takéhoto chybového vektoru s dĺžkou *n* je základným princípom fungovania systémov založených na QC-MDPC systéme, pretože po prenásobení správy maticou G dostávame kódové slovo, ktoré má na prvých *k*pozíciách práve pôvodnú správu. Preto je potrebné správu maskovať a to pripočítaním chybového vektoru.

V binárnych QC-MDPC kódoch sa tento chybový vektor odstraňuje dešifrovaním pomocou bit-flippingových algoritmov [3][8], ktorých podstata spočíva v iteratívnom hľadaní bitov zašifrovanej správy, ktoré najviac prispievajú k nenulovému syndrómu tejto správy.

#### Definícia 7: Syndróm správy

*Majme lineárny kód C popísaný kontrolnou maticou H a majme kódové slovo*

*c ∈ C. Potom platí:*

cHT = 0,

*kde 0 je nulový vektor s r = n – k prvkami. Tento vektor nazývame nulovým syndrómom správy.*

Z predchádzajúcej definície vyplýva, že každé slovo, ktoré je kódovým slovom daného lineárneho kódu, má nulový syndróm správy, ktorý dostaneme prenásobením tohto kódového slova transponovanou maticou H. Ak slovo nie je kódovým slovom lineárneho kódu, syndróm správy bude nenulový.

Práve spomínané bit-flippingové algoritmy využívajú syndróm správy na odhad, ktoré pozície zašifrovanej správy prispievajú do nenulového syndrómu najviac a teda boli pravdepodobne ovplyvnené pridaním chybového vektoru.

V článku [3] boli prezentované sofistikovanejšie metódy odhaľovania chybového vektoru s použitím tzv. soft-decoding techník, ktoré umožnili dekódovanie s ešte menšou chybou dekódovania a tak zvýšiť bezpečnosť asymetrického systému s použitím lineárneho kódu s rovnakými parametrami.

Práve nízka chyba dekódovania v kombinácii s kompresiou kľúčov je hlavným predmetom výskumu možného použitia QC-MDPC kódov. V článku [6] bol prezentovaný systém založený na QC-MDPC kódoch, avšak s použitím nebinárnej abecedy. Autori tohto systému deklarujú ešte menšie kľúče so zachovaním dostatočne malej chyby dekódovania. Použitie dostatočne veľkej abecedy taktiež prináša nové dekódovacie techniky, ktoré nie sú realizovateľné v kódoch s binárnou veľkosťou abecedy.

# QC-MDPC nad GF(P)

QC-MDPC systém sa stal atraktívnym pre výskum najmä kvôli výraznej redukcii verejného kľúču oproti originálnemu McEliecovmu systému založenom na Goppa kódoch. Nevýhodou a hlavnou zraniteľnosťou tohto systému voči spomenutému systému je však dekódovanie, keďže iteratívne dekódovanie kódov so stredne hustou maticou môže zlyhať s určitou pravdepodobnosťou. Táto pravdepodobnosť závisí od nastavených parametrov použitého systému a to najmä od Hammingovej váhy pridaného chybového vektoru v binárnej verzií QC-MDPC kódov.

V práci [6] je navrhnutá verzia systému s použitím nebinárnej abecedy v QC-MDPC kódoch, ktorá podľa autorov umožňuje ešte viac redukovať veľkosť verejného kľúču a to pri zachovaní porovnateľnej úrovni bezpečnosti. Použitie nebinárnej abecedy taktiež prináša nové techniky iteratívneho dekódovania takýchto kódov.

## Popis QC-MDPC kódov nad GF(P)

Vzhľadom na fakt, že návrh s použitím nebinárnej abecedy je rozšírením QC-MDPC systémov založených na binárnej abecede, schéma je veľmi podobná pôvodnej schéme. Najvýznamnejším rozdielom je použitie nových dekódovacích techník, ktoré budú popísané v kapitole 4. V priebehu času sa táto schéma zjednodušovala, avšak pre objasnenie toho, ako vznikala výsledná schéma, ktorá bola ako súčasť tejto Diplomovej práce implementovaná, je dobré uviesť prehľad týchto návrhov.

### Konštrukcia MDPC kódov nad GF(P) [6]

1. Vygenerujeme r vektorov dĺžky n: (hi ∈ Fpn)0≤i<r, každý z vektorov obsahuje wsig pozícii w1 s hodnotou v rozpätí {-1, 1} a w2 pozícii s hodnotou z množiny {-2, 2} nad poľom GF(P). Ostatné pozície budú obsahovať hodnotu 0.
2. Výsledná kontrolná matica H ∈ Fpr×n definuje prislúchajúci MDPC kód nad GF(P).

Z vyššie uvedených bodov je teda jasná riedkosť matice H, ktorá závisí od nastavenia w1, respektíve w2. Matica bude obsahovať iba 4 hodnoty rôzne od nuly a teda ak by sme nastavili hodnotu P = 7, matica nad GF(7) by obsahovala w1 prvkov z množiny {1, 6} a w2 prvkov z množiny {2, 5}. Ostatné pozície matice by znovu obsahovali hodnotu 0.

### Konštrukcia QC-MDPC kódov nad GF(P) [6]

1. Vygenerujeme vektor h dĺžky n: (h ∈ Fpn).Vektor obsahuje wsig pozícii w1 s hodnotou v rozpätí {-1, 1} a w2 pozícii s hodnotou z množiny {-2, 2} nad poľom GF(P). Ostatné pozície budú obsahovať hodnotu 0.
2. Výsledná kontrolná matica H ∈ Fpr×n, s prvým riadkom identickým vektoru h, definuje prislúchajúci QC-MDPC kód nad GF(P).
3. Zostávajúcich r – 1 riadkov môžeme zostrojiť r – 1 kvázi-cyklickými posunmi prvého riadku
4. Vzhľadom na vlastnosti kvázi-cyklických matíc, konštruujeme kontrolnú maticu H ako : H = [H0 | H1 | ... | Hn0 - 1].
5. Každý z blokov Hi je izomorfný k polynómu: hi(x) ∈ R = Fp[x]/ (xr – 1).

Z uvedených popisov schém je možné vidieť jediný rozdiel medzi QC-MDPC a MDPC kódmi a to generovanie matice iba pomocou prvého riadku. Priamym dôsledkom takejto konštrukcie matíc je výrazná pamäťová úspora.

Z takto zostrojenej kontrolnej matice H, je možné zostrojiť generujúcu maticu G, ktorá bude rovnako popísaná svojim prvým riadkom. Tento riadok bude znovu pozostávať z viacerých blokov izomorfných k polynómom v rovnakom zvyškovom okruhu ako polynómy reprezentujúce prvý riadok matice H. Parameter systému *r* by mal podľa článku predstavovať prvočíslo.

Autori však svoj pohľad na zostrojenie takéhoto QC-MDPC kódu nad GF(P) mierne zmenili. Preto si uvedieme aj ďalšiu schému takéhoto kódu, avšak podľa novšieho článku.

## Schéma QC-MDPC kódov nad GF(P) [7]

Schéma konkrétneho kryptosystému, ktorý bol implementovaný ako súčasť tejto Diplomovej práce si uvedieme v tejto kapitole. Od pôvodného článku [6] sa systém líši najmä v zostrojení kontrolnej matice H a voľbe veľkosti konečného poľa GF(P).

Kvázi-cyklická kontrolná matica H je zostavená z 2 cyklických blokov H0 a H1 rozmeru (k x k). Tieto matice sú jednoznačne popísané prvým riadkom h0 resp. h1 dĺžky *r*. Zostávajúce riadky blokov sú zostrojené (k – 1) cyklickými posunmi prvých riadkov. Postup pre zostrojenie týchto vektorov si uvedieme v nasledujúcom odseku.

#### Zostrojenie kontrolnej matice H a generujúcej matice G:

Pre potreby generovania kľúčov potrebujeme definovať niekoľko parametrov použitých v tomto procese. Vyššie spomenutý parameter k vyjadruje rozmer cyklických blokov, z ktorých sa skladajú matice kvázi-cyklického kódu. Veľkosť abecedy nám v tomto konkrétnom návrhu určuje parameter q. Autori návrhu odporúčajú voliť kombinácie parametrov k a q nasledovne:

q – mocnina čísla 2

k – prvočíslo

Zq - definuje zvyškový okruh.



q – prvočíslo

k – mocnina čísla 2

Zq – definuje konečné pole

V oboch prípadoch Zqkdefinuje rozšírenie konečného poľa, resp. zvyškového okruhu na k pozícii, ktoré je definované ako zvyškový okruh po delení polynómom p(x) = xk – 1. Označujeme:

R = Zq [X] / <Xk – 1>.

Parameter d môžeme zatiaľ chápať ako pozitívne celé číslo, ktoré má vplyv na rozsah čísel, z ktorých sú generované prvky matice H. Prvým krokom je zostrojenie vektorov h0ˆ a h1ˆ, pre ktoré platí:

hiˆ ← Idk,

kde Id = {-d, … , 0 , … d} a suma prvkov vektoru hiˆ je rovná 0.

Ak teda pre zjednodušenie budeme uvažovať P = 7 a d = 1, všetky prvky vektoru h­iˆ budú z množiny Id = {-6, 0, 6}. Vzhľadom na nulovú sumu prvkov vektoru tiež vyplýva fakt, že počet prvkov s hodnotou -6 a 6 bude rovnaký.

Ďalším krokom je výpočet čísel a výsledných vektorov, ktorých prvé pozície závisia od týchto hodnôt.

h0d = q / (2\*d + 1),

h1d = q / (2\*d + 1)2.

vektor h0 = h0d (1, 0, ..., 0)k + h0ˆ,

vektor h1 = h1d (1, 0, ..., 0)k + h1ˆ.

Z prvých riadkov môžeme zostaviť cyklické matice H0 a H1­. Ak je matica H1 singulárna, je potrebné toto generovanie zopakovať. Zoradením matíc H0 a H1 dostávame kvázi-cyklickú maticu H. Maticu G môžeme zostrojiť ako

G = (I | H1-1 H0),

kde I je jednotková matica.

Generujúca matica G teda zostáva v systematickej forme a tak po násobení správy touto maticou dostávame kódové slovo, ktoré ma na prvých k pozíciách pôvodnú správu, za ktorou nasleduje redundantná časť, ktorá zohráva úlohu pri dekódovaní zašifrovanej správy spätne na kódové slovo a to pomocou hľadania nulového syndrómu správy.

**Šifrovanie**

1. Nech správa m ∈ Fqk je otvorený text a chybový vektor e ← Idn. Chybový vektor teda obsahuje n prvkov. Každá z pozícii vektora e je rovnomerným rozdelením vygenerovaná z množiny Id.
2. Po šifrovaní dostávame zašifrovaný text c ∈ Fqn. Platí c = m \* G + e.

**Dešifrovanie**

1. Nech správa c ∈ Fqn je zašifrovaná správa. Využitím vlastnosti lineárnych kódov a redundancie tohto kódu, vieme v dekódovacom procese extrahovať chybový vektor e a to tak, že sa zašifrovaná správa po odčítaní tohto vektoru stane kódovým slovom a bude tak mať nulový syndróm.
2. Pôvodnú správu potom vieme odčítať z prvých k pozícii takto dešifrovanej správy.

# Dekódovanie P-ary QC-MDPC kódov

Použitie nebinárnej abecedy prináša nové dekódovacie techniky pre QC-MDPC kódy nad GF(P). Zatiaľ, čo pri použití binárnej abecedy spočíva hľadanie pôvodnej správy v preklápaní bitov, ktoré prispievajú do nenulového syndrómu správy najväčšou mierou, v novej OQ-MDPC P-ary schéme je možné využiť veľkosť použitej abecedy. Najskôr si uvedieme Algoritmus 1 z článku [7]

## Algoritmus 1 [7]

Podľa autorov článku, tento algoritmus vychádza z bitflippingového algoritmu z článku [11] a je úpravou tohto algoritmu na potreby dekódovania QC-MDPC kódov nad GF(P). Pôvodný algoritmus bol pôvodne navrhnutý pre lattice-based kryptosystém.

Algoritmus berie ako vstup zašifrovanú správu *c*, dĺžky n a kontrolnú maticu *H* s rozmerom (k x n) a parameter *iter*, ktorý špecifikuje počet iterácii pre dekódovací algoritmus. Graphical user interface, text, application, email

Description automatically generated

Tento algoritmus vráti chybový vektor *e*, ak bolo dekódovanie úspešné, alebo vráti ⊥ - chybu dekódovania – ak počas behu zadaného počtu iterácii nebolo možné odhaliť chybový vektor tak, aby bol syndróm vypočítaný na riadku 7 nulovým syndrómom.

Myšlienka algoritmu spočíva v iteratívnom upravovaní pozícii chybového vektoru podľa pravidiel implementovaných v metóde Decide, ktorá je volaná na riadku 5. Ak bol nájdený skutočný chybový vektor, dostávame na riadku 8 nulový vektor. Do pôvodného syndrómu správy totiž prispieval len tento vektor (syndróm kódového slova je nulový vektor), takže po odčítaní vektoru, ktorý dostávame po vynásobení H \* eT od pôvodného syndrómu, dostávame nulový syndróm správy. V prípade, že v žiadnej z iterácii nenachádzame nulový syndróm, nastáva chyba dekódovania a algoritmus nebol schopný odhaliť pôvodné kódové slovo, ktoré vzniklo systematickým kódovaním z otvoreného textu *m* a generujúcej matice *G,­* ktorá je v systematickom redukovanom tvare.

Najdôležitejšou časťou algoritmu zostáva súbor rozhodovacích pravidiel v metóde Decide, ktorý počas iterácii dekódovacieho algoritmu vplýva na chybový vektor. Rozhodovacie pravidlá sú postavené na vlastnosti kontrolnej matice H. Táto matica vo svojom prvom riadku totiž obsahuje 2 vektory (reprezentujúce polynómy), každý s jednou významnou pozíciou.

V kapitole 3.2 sme uviedli, že výpočet vektorov a ich prvých pozícii vyzerá nasledovne:

h0 = h0d (1, 0, ..., 0)k + h0ˆ,

h1 = h1d (1, 0, ..., 0)k + h1ˆ.

kde

h0d = q / (2\*d + 1),

h1d = q / (2\*d + 1)2.

Pre jednoduchosť môžeme uvažovať parameter d = 1, teda Id = {-1, … , 0 , … 1}. Pri výpočte prvých pozícii teda treba brať v úvahu, že na prvej pozícii vektora h0 je hodnota h0d sčítaná s prvou pozíciou vektora h0ˆ , respektíve na prvej pozícii vektora h1 je hodnota h1d sčítaná s prvou pozíciou vektora h1ˆ.

#### Príklad 3: Zostrojenie prvého riadku matice H

O matici H môžeme uvažovať ako o matici, ktorá je zložená z 2 cyklických blokov v horizontálnom smere. Preto vieme o tejto matici povedať, že na jej skonštruovanie nám stačí poznať prvý riadok dĺžky n, ktorý obsahuje 2 vektory dĺžky k. Sú to vektory h0 a h1. Potom k – 1 kvázi-cyklickými rotáciami tejto matice vieme zostrojiť celú kontrolnú maticu.

Pre jednoduchosť uvažujme d = 1, k = 4, q = 23. Potom výpočet prvého riadku matice môže vyzerať nasledovne:

h0d = 23 / (2\*1 + 1) = 8,

h1d = 23 / (2\*1 + 1)2 = 3,

h0ˆ = [0, 1, -1, 0],

h1ˆ = [1, -1, 0, 0],

h0 = h0d [1, 0, 0, 0] + h0ˆ = 8\*[1, 0, 0, 0] + [0, 1, -1, 0] = [8, 1, -1, 0],

h1 = h1d [1, 0, 0, 0] + h1ˆ = 3\*[1, 0, 0, 0] + [1, -1, 0, 0] = [4, -1, 0, 0].

Vidíme, že hodnoty h0d a h1d sú získané štandardným zaokrúhlením jednoduchého výpočtu do ktorého vstupuje parameter *q* a *d*. Vektory h0ˆ a h1ˆ sú vektory dĺžky k = 4 a každá pozícia týchto vektorov pochádza z generovania rovnomerným rozdelením z množiny Id = {-1, 0, 1}. Preto dostávame 2 významné pozície v prvom riadku matice a to je pozícia h0(1), ktorá v tomto príklade predstavuje prvý prvok vektoru h a pozícia h1(1), ktorá predstavuje piaty prvok vo vektore h. Pre lepšiu ilustráciu, vektor h má podobu:

h = [h­0, ­h1] = [**h0(1),** h0(2), h0(3), h0(4), **h1(1),** h1(2), h1(3), h1(4)],

kde významné pozície sú vyznačené hrubým písmom. Tieto pozície zohrávajú významnú úlohu pri dekódovaní, pretože pri správne zvolených parametroch k a q – rádovo v stovkách – sú to jediné pozície ktoré obsahujú hodnoty rôzne od hodnôt z množiny Id.

#### Algoritmus Decide

V algoritme 1 na riadku 5 je volaná metóda Decide. Táto metóda predstavuje rozhodovací algoritmus, ktorý upravuje chybový vektor na základe súboru pravidiel. Tieto pravidlá sú odvodené z štruktúry kontrolnej matice H. Kontrolná matica je popísaná prvým riadkom h, ktorý obsahuje 2 významne pozície rôzne od -1, 0, 1.

Každý prvok chybového vektora je opäť z hodnôt -1, 0, 1. Uvažujeme teda, že pri výpočte syndrómu do každej pozície tohto syndrómu najviac prispievajú významné pozície **h0(1)** a **h1(1)**. Keďže chybový vektor na každej pozícii môže nadobúdať iba 3 hodnoty, tak pozície chybového vektora násobené významnými pozíciami maticami matice H môžu byť iba v 9 kombináciách. Pre lepšiu ilustráciu si uvedieme príklad výpočtu prvej pozície syndrómu správy.

#### Príklad 4: Výpočet syndrómu správy

Uvažujme riadkový chybový vektor *e* dĺžky n, kde n = 2 \* k. Uvažujme stĺpcový vektor *s*, ktorý predstavuje syndróm správy. Správa vznikla pripočítaním chybového vektoru ku kódovému slovu, a teda do syndrómu správy prispieva iba tento chybový vektor. Každá z pozícii chybového vektora je rovnomerne náhodne vybratá z množiny Id = {-d, ... 0..., d}.

Pre jednoduchosť môžeme zvoliť parameter d = 1. Potom prvý riadok matice H má dve významné pozície s približnými hodnotami: h0 = q / 3, hk+0= q / 9. Na základe pravidiel generovania prvého riadku matice H je možné, že tieto významné pozície budú zmenšené, resp. zväčšené o hodnotu z Id. Prvý riadok matice má nasledovnú štruktúru:

h = [q/3 ... q/9 ...]

Pri dostatočne veľkom q sú tieto významné pozície výrazne odlišné od ostatných pozícii a vieme vďaka ním odhadovať pôvodný chybový vektor. Ten zisťujeme pomocou syndrómu. Každá pozícia syndrómu vznikne vynásobením riadku matice H s prislúchajúcim indexom a transponovaným chybovým vektorom eT.

s = H \* eT.

Takto vieme vypočítať každú pozíciu syndrómu s. Príklad výpočtu pozície s­0:

s0 = h0 \* eT = (q / 3) \* eT0 + (q / 9) \* eTk+0 + šum,

kde šum môžeme vyjadriť ako sumu zostávajúcich členov hi \* eTi, kde i je rôzne od 0 a rôzne od k. Šum teda predstavuje sumu súčinov členov e­­Ti a hi, kde tieto členy nadobúdajú hodnoty z Id. Ich súčiny teda rovnako nadobúdajú hodnoty z Id, avšak nie podľa rovnomerného rozdelenia.

#### Tabuľka 1: Súčiny členov e­­Ti a hi, kde i je rôzne od 0 a rôzne od k

|  |  |  |
| --- | --- | --- |
| eTi | hi | hi \* eTi |
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | -1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |
| 1 | -1 | -1 |
| -1 | 0 | 0 |
| -1 | 1 | -1 |
| -1 | -1 | 1 |

Z tabuľky môžeme vidieť, že hodnota súčinu týchto členov nadobúda hodnotu 0 s najväčšou pravdepodobnosťou. Taktiež vzhľadom na rovnomernosť generovania prvkov vektorov *h* a *e*, môžeme prehlásiť, že súčiny s hodnotou 1 a -1 sa vyskytujú rovnako často. Preto môžeme tvrdiť, že šum je súčet hodnôt -1, 0, 1, kde hodnota 0 má najväčšiu pravdepodobnosť výskytu na danej pozícii, a 1 a -1 má rovnakú pravdepodobnosť výskytu na danej pozícii.

Z predchádzajúceho príkladu vidíme, že najväčší vplyv na pozíciu syndrómu majú pozície eTi a eTi+0. Keď zoberieme do úvahy fakt, že každá pozícia chybového vektoru je z množinu Id = {-1, 0, 1}, dostávame 9 možných kombinácii, ktoré si uvedieme v tabuľke. Týchto 9 pozícii môžeme chápať ako karteziánsky súčin Id x Id.

#### Tabuľka 2: Odhad hodnôt pozícii syndrómu s

|  |  |  |
| --- | --- | --- |
| ei | ei+k | si = e­i \* q/3 + eí + k \* q/9 |
| 0 | 0 | 0 + 0 = 0 |
| 0 | 1 | 0 + 1/9q = 1/9q |
| 0 | -1 | 0 – 1/9q = 8/9q |
| 1 | 0 | 1/3q + 0 = 3/9q |
| 1 | 1 | 1/3q + 1/9q = 4/9q |
| 1 | -1 | 1/3q – 1/9q = 2/9q |
| -1 | 0 | -1/3q + 0 = 6/9q |
| -1 | 1 | -1/3 + 1/9q = 7/9q |
| -1 | -1 | -1/3q – 1/9q = 5/9q |

Na základe hodnoty pozície syndrómu si vieme približne odhadnúť hodnoty ei a ek+i. Tieto intervaly si vieme znázorniť na kružnici.

#### Obrázok 1: Intervaly pre výpočet pozícii chybového vektoru

A picture containing diagram

Description automatically generated

Na základe tohto obrázku vieme určiť hranice pre rozhodovací algoritmus. Stále uvažujeme d = 1 a teda máme 9 signálnych bodov, pretože Id = {-d, 0, d} = {-1, 0, 1}. Intervaly pre rozhodovanie teda môžeme vyčítať z nasledujúcich obrázkov.

Obrázok 2: Intervaly pre významnú pozíciu **h0(1): [7]**

A picture containing electronics

Description automatically generated

Z obrázku môžeme vyčítať, že ak je hodnota si v rozmedzí od [q/6, q/2], môžeme odhadnúť ei na 1. V prípade, že s­­i spadá do [q/2, 5q/6], môžeme o ei prehlásiť že najlepším odhadom je -1 pre túto pozíciu chybového vektoru. V poslednom intervale [5q/6, q/6] môžeme e­i odhadnúť ako 0.

Rovnako ako obrázok 2 môžeme popísať aj nasledujúci obrázok pre odhad e­k+i. Odvodenie hraníc pre odhad sa však mierne líši od predchádzajúceho prípadu a preto si ho popíšeme bližšie.

Obrázok 3: Intervaly pre významnú pozíciu h1**(1): [7]**

Diagram

Description automatically generated

Vidíme, že parameter *q* a prislúchajúce hranice – frakcie jeho hodnoty – vystupujú v tomto prípade zredukované na tretinu veľkosti ako to bolo v prípade odhadu ei. Pre odôvodnenie sa môžeme pozrieť naspäť do obrázku 1 a tabuľky 2, kde vidíme, že hodnota pre ei+k­­­ sa opakuje s periódou rovnou 3. Teda ak zredukujeme hranice na tretinu, vieme nájsť súvislý interval pre odhad hodnoty ei+k­.

#### Príklad 5: Úprava intervalu pre odhad hodnoty ei+k:

Pozrime sa do tabuľky 2 a nájdime ei+k rovné 0. Dostávame tabuľku:

|  |  |  |  |
| --- | --- | --- | --- |
| ei | ei+k | si | si mod q/3 |
| 0 | 0 | 0 | 0 |
| 1 | 0 | 3/9 \* q | 1/9 \* q |
| -1 | 0 | 6/9 \* q | 2/9 \* q |

Ak si zredukujeme mod q/3, dostaneme kružnicu s rozdelením intervalov ako na *Obrázku 3*. Takto upravené intervaly nám v dekódovacom algoritme umožnia jednoduchý odhad pre hodnotu chybového vektoru ei+k­.

Po opísaní rozhodovacieho algoritmu Decide si môžeme uviesť jeho pseudokód, kde môžeme vidieť aplikovanie vyššie popísaných pravidiel.

#### Algoritmus Decide [7]

Algoritmus berie ako vstup vektor *p*, ktorý predstavuje v tomto prípade syndróm správy. Výpočet syndrómu je výsledkom násobenia transponovaného chybového vektoru a matice H tak, ako bolo spomenuté v predchádzajúcom odseku. Vidíme, že rozhodovací algoritmus v cykle prechádza všetky prvky syndrómu.

Graphical user interface, text, application

Description automatically generated

Od riadku 3, po riadok 7 prebieha kontrola hraníc intervalov pre rozhodnutie o pôvodnej hodnote chybového vektora e­j, teda pozeráme na obrázok 2 a to takým spôsobom, že vyberáme interval, do ktorého spadla hodnota pozície syndrómu pj. Obdobne postupujeme v druhej časti chybového vektoru *e*.

Od riadku 8 po riadok 12 prebieha kontrola hraníc intervalov pre rozhodnutie o pôvodnej hodnote chybového vektoru e­j+k, teda pozeráme na obrázok 3 a to takým spôsobom, že vyberáme interval, do ktorého spadla hodnota pozície syndrómu pj mod q/3.

Po prechode cez celý syndróm správy je vrátený upravený chybový vektor *e*, z ktorého bude v nasledujúcej iterácii Algoritmu 1 vypočítaný syndróm, ktorý ak bude stále nenulový, prejde znovu do Algoritmu Decide novej iterácie dekódovacieho Algoritmu 1.

Zovšeobecneným algoritmu 1 a rozhodovacieho algoritmu Decide je algoritmus DecideThr, ktorý upravuje intervaly počas rozhodovania o prahovú hodnotu *t­­hreshold.* Takto definovaný algoritmus je možné parametrizovať viacerými metódami. Jednou z možností je upravovanie *thresholdu* vo viacerých iteráciách.

#### Algoritmus DecideThr [7]

Do algoritmu znovu vstupuje syndróm správy označený v tomto prípade ako vektor *p.* Podobne ako pri Decide, prechádzame prvky syndrómu *p* a upravujeme chybový vektor tak, aby korešpondoval s rozhodovacími pravidlami. Tieto rozhodovacie pravidlá sú však mierne pozmenené a to tak, že hranice intervalov sú parametrizovateľné o hodnotu *thr(i).* Hodnota tohto parametru má za následok zmenšenie intervalu pre rozhodovanie z oboch strán.

Text

Description automatically generated

V prípade ak určíme hodnotu *thr(i)* rovnú nule, dostávame z algoritmu DecideThr algoritmus Decide. Taktiež môžeme použiť inú hodnotu pre *threshold* v závislosti od aktuálnej iterácie. Takto môžeme začať s malým intervalom pre rozhodovanie o nenulovej hodnote v chybovom vektore a postupne tento interval rozširovať až pokiaľ nedosiahneme nulový syndróm, alebo kým nenastane chyba dekódovania.

#### Obrázok 4: Parametrizovanie algoritmu DecideThr v iteráciách [7]

Diagram

Description automatically generated with medium confidence

Ako môžeme vidieť z obrázku 4, prvá iterácia algoritmu DecideThr začne s najväčším prahom, čo znamená najužší interval pre rozhodovanie o nenulových hodnotách spomedzi všetkých iterácii. V poslednej iterácii, ak doposiaľ nebol nájdený chybový vektor, ktorého syndróm zodpovedá syndrómu zašifrovanej správy, je *threshold* najmenší a preto sú intervaly pre rozhodovanie o nenulových hodnotách najväčšie.

Takto definovaný algoritmus DecideThr je možné použiť a parametrizovať s využitím rôznych metód pre výpočet syndrómu. Príkladom je heuristický dekóder z článku [7].

## Heuristický dekóder – TODO

Princíp dekódovania heuristickým dekóderom zostáva rovnaký ako pri dekódovaní Algoritmom 1. Jedinou zmenou je použitie vyššie spomenutého algoritmu DecideThr namiesto použitia algoritmu Decide. Keďže sa v algoritme DecideThr v každej iterácii prepočítava threshold, rozsah intervalu pre rozhodovanie o hodnotách pozícii chybového vektoru nie je statický, ale môže sa v každej iterácii meniť.

# Návrh riešenia a testovanie

Zadanie bolo implementované v jazyku C/C++ s použitím knižníc NTL a FLINT. Implementácia sa nachádza v projekte DiplomaThesis, ktorý v hlavnom programe Main v súbore main.cpp v koreňovom adresári volá základne testy pre otestovanie implementácie p-ary QC-MDPC kryptosystému.

V priečinku Cipher sa nachádzajú implementácie dešifrovacích algoritmov v súbore decipher.cpp, respektíve šifrovacieho algoritmu v súbore encipher.cpp. V prislúchajúcich hlavičkových súboroch sú tieto algoritmy deklarované.

V priečinku Context sa nachádzajú triedy GeneratorMatrix, ParityCheckMatrix, PlainText, CipherText, ktoré sú výstupmi a vstupmi vyššie spomenutých algoritmov.

Kľúče sú generované pomocou metód deklarovaných v hlavičkovom súbore key\_generation.h v priečinku KeyGeneration. V spomenutom hlavičkovom súbore sa taktiež nachádzajú definície konštánt pre parametrizovanie QC-MDPC p-ary kryptosystému.

Testovanie bolo uskutočnené s použitím jedného jadra procesora Intel CORE vPro i7 11. generácie.

## Dekódovacia chyba Algoritmu 1

V tejto sekcii sú zhrnuté výsledky testovania implementovaného Algoritmu 1 a porovnané voči hodnotám uvedeným v práci [7] pre Algoritmus 4. Porovnávaná je hodnota DFR, teda relatívna chybovosť dekódovania pre oba algoritmy. Experimenty boli v tejto práci vykonávané pre 100 000 dekódovaní, pričom bolo pre 1000 náhodne vygenerovaných inštancii kľúčov pre šifrovanie a dešifrovanie vykonaných 100 zašifrovaní náhodných správ a následne ich spätných dešifrovaní pomocou výpočtu chybového vektora. Experiment pre Algoritmus 4 bol v práci [7] vykonaný pre 10 000 000 dekódovaní, pričom pre každú z 10 000 QC-MDPC inštanciu bolo vykonaných 1000 dekódovaní náhodných správ.

#### Tabuľka 3: Dekódovacia chyba Algoritmu 1, pri k = 491, q = 345

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| k | q | Iter | Algoritmus 4 [7] | Algoritmus 1 | Čas dekódovania algoritmom 1 |
| 491 | 345 | 10 | - | 24 213 | 0,1414 s |
| 491 | 345 | 15 | - | 870 | 0,1505 s |
| 491 | 345 | 20 | - | 273 | 0,1508 s |
| 491 | 345 | 25 | 145 | 24 | 0,1532 s |
| 491 | 345 | 30 | - | 10 | 0.1527 s |
| 491 | 345 | 35 | - | 12 | 0,1524 s |
| 491 | 345 | 40 | - | 8 | 0.1598 s |
| 491 | 345 | 45 | - | 2 | 0.1572 s |
| 491 | 345 | 50 | 10 | 0 | 0,1551 s |
| 491 | 345 | 75 | 4 | 0 | 0,1547 s |
| 491 | 345 | 100 | 2 | 0 | 0,1544 s |

V Tabuľke 3 môžme vidieť v stĺpci *Algoritmus 4* odhadovaný počet chýb, ktoré urobí tento algoritmus pri pri dekódovaní 100 000 správ za vyššie uvedených podmienok. V stĺpci *Algoritmus 1* môžme vidieť počet chýb, ktoré urobil algoritmus implementovaný ako súčasť tejto diplomovej práce pri 100 000 dekódovaniach. V stĺpci *Čas dekódovania algoritmom 1* môžme vidieť, ako dlho trvalo v priemere jedno dekódovanie správy pre daný počet iterácii, bez ohľadu nato, či bolo dekódovanie úspešné.

#### Graf 1: Závislosť chyby dekódovania – DFR - od počtu iterácii dekódovania

Na *Grafe 1* je znázornená DFR – Decode Failure Rate – pre *Algoritmus 1*. Pre počet iterácii väčší ako 45 sme neboli schopní namerať dekódovaciu chyby, preto nie je na logaritmickej škále vynesený žiadny bod, môžme však prehlásiť, že nami odhadovaná dekódovacia chyba pri takto nastavených iteráciách je menšia ako 10-5.

#### Graf 2: Porovnanie chybovosti Algoritmu 1 a Algoritmu 4 [7] pre 100 000 dekódovaní

Z Grafu 2 môžme vyčítať, že Algoritmus 1 implementovaný v tejto práci dosiahol v porovnaní s výsledkami prezentovanými v práci [7] menšiu DFR. Pri 100 00 dekódovaniach a počte iterácii dekódovacieho algoritmu 25, dosiahol 6-násobne menšiu DFR ako Algoritmus 4. Pri počte iterácii 50 a viac sme pri 100 000 dekódovaniach sme nedosiahli žiadne chybné dekódovanie, preto môžme odhadnúť DFR na hodnotu menšiu ako 10-5, čo je znovu menej ako pri Algoritme 4.

#### Tabuľka 4: Dekódovacia chyba Algoritmu 1, pri k = 491, q = 360 pre 100 000 dekódovaní

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| k | q | Iter | Algoritmus 4 [7] | Algoritmus 1 | Dekódovanie algoritmom 1 |
| 491 | 360 | 10 | - | 409 | 0,1401 s |
| 491 | 360 | 25 | - | 11 | 0,1422 s |
| 491 | 360 | 20 | - | 4 | 0,1435 s |
| 491 | 360 | 25 | 2,67 | 0 | 0,1472 s |
| 491 | 360 | 30 | - | 0 | 0.1445 s |
| 491 | 360 | 35 | - | 0 | 0,1439 s |
| 491 | 360 | 40 | - | 0 | 0.1468 s |
| 491 | 360 | 45 | - | 0 | 0.1436 s |
| 491 | 360 | 50 | 0,17 | 0 | 0,1438 s |
| 491 | 360 | 75 | 0,04 | 0 | 0,1429 s |
| 491 | 360 | 100 | 0,05 | 0 | 0,1436 s |

Pri nastavení parametru q na hodnotu 360 sme namerali nižšie hodnoty DFR ako pri q rovné 345. Rovnako ako v *Tabuľke 3* môžme vidieť, že nami implementovaný *Algoritmus 1* dosahoval v priemere nižšiu DFR ako *Algoritmus 4­* z článku [7]. Pri počte iterácii 25, sme nenamerali žiadne chybné dekódovanie, preto DFR môžme znovu odhadnúť na hodnotu menšiu ako 10-5, pričom *Algoritmus 4­­­* zlyhal pri 100 000 dekódovaniach v priemere 2,67 krát.

## Dekódovacia chyba Heuristického dekódera - TODO

## Porovnanie veľkosti kľúčov pre 128 bitovú bezpečnosť BIKE vs p-ary QC-MDPC

# **Záver**

Cieľom tejto Diplomovej práce bola implementácia QC-MDPC kryptosystému s nebinárnou abecedou podla článkov [6] a [7] a taktiež implementácia vybraných dekódovacích algoritmov a ich následne testovanie. Výsledkom tejto práce je úspešná implementácia takéhoto systému s dekódovacím algoritmom v práci označovaným ako *Algoritmus 1.* Pre tento dekódovací algoritmus bola taktiež nameraná dekódovacia chyba DFR a porovnaná s referenčnými hodnotami pre *Algoritmus 4* z článku [7].

Z výsledkov testovania uvedených v kapitole 5 môžme usúdiť, že *Algoritmus 1* implementovaný ako súčasť tejto diplomovej práce, dosahoval pre rovnaké parametre k, q a počet iterácii dekódovania v priemere nižšie hodnoty DFR ako *Algoritmus 4,* ktorý bol v článku [7] uvádzaný ako bit-flippingový dekóder pre lattice-based protokol na výmenu kľúčov Ouroboros-E [11].

# **Zoznam použitej literatúry**

[1] MCELIECE R. J. ”A public-key cryptosystem based on algebraic coding theory.”

Dostupné na internete: https://tmo.jpl.nasa.gov/progress\_report2/42-44/44N.PDF

[2] SHOR, P. W. „Algorithms for quantum computation: discrete logarithms and

factoring“. In Proceedings 35th Annual Symposium on Foundations of Computer

Science. 1994, s. 124–134. ISBN Print ISBN: 0-8186-6580-7. Dostupné na internete: https://ieeexplore.ieee.org/document/365700

DOI: 10.1109/SFCS.1994.365700.

[3] ARAGON N. et al. „BIKE: Bit Flipping Key Encapsulation“. 2019. Dostupné na

internete: <https://bikesuite.org/files/v4.0/BIKE_Spec.2020.05.03.1.pdf>

NIST. Post-Quantum Cryptography Standardization [online] [cit. 2020-05-28].

Dostupné na internete: <https://nvlpubs.nist.gov/nistpubs/ir/2016/NIST.IR.8105.pdf>

[5] https://csrc.nist.gov/News/2022/pqc-candidates-to-be-standardized-and-round-4

[6] Guo Q., Johanson T.: „A p-ary MDPC scheme“ . 2016 IEEE

[7] Guo Q., Johanson T., Martinez I.: „Improved Decoders for p-ary MDPC“. 2019

[8] QC-MDPC

[9] KRAVECOVÁ D. „Základy kódovania“. FEI TU, Košice, 2012, s 67-76. ISBN: 978-80-553-1178-4

[10] KOVÁČ, J. „Reakčný útok na QC-MDPC McEliece kryptosystém“. Slovenská

technická univerzita v Bratislave, Fakulta elektrotechniky a informatiky. 2019.

Diplomová práca

[11] Deneuville, J., Gaborit, P., Guo, Q., Johansson, T.: Ouroboros-e: An efficient

lattice-based key-exchange protocol. In: 2018 IEEE International Symposium on

Information Theory, ISIT 2018, Vail, CO, USA, June 17-22, 2018. pp. 1450–1454.

IEEE (2018), <https://doi.org/10.1109/ISIT.2018.8437940>

[12] ROSENTHAL C. M. J. – SHOKROLLAHI A. ”Using low density parity check codes in the McEliece cryptosystem.”, 2000 IEEE. Dostupné na internete: DOI: 10.1109/ISIT.2000.866513

# **Prílohy**

[1] Implementácia p-ary QC-MDPC kryptosystému s použitím NTL a FLINT knižníc v jazyku C / C++