

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Evidenčné číslo: FEI-5382-72989

QC-MDPC MCELIECE NA MIKROPROCESORE
BAKALÁRSKA PRÁCA

2016

Radovan Bezák

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY**

Evidenčné číslo: FEI-5382-72989

**QC-MDPC MCELIECE NA MIKROPROCESORE
BAKALÁRSKA 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: doc. Ing. Pavol Zajac, PhD.

Bratislava 2016

Radovan Bezák



ZADANIE BAKALÁRSKEJ PRÁCE

Študent: **Radovan Bezák**
ID študenta: 72989
Študijný program: Aplikovaná informatika
Študijný odbor: 9.2.9. aplikovaná informatika
Vedúci práce: doc. Ing. Pavol Zajac, PhD.
Miesto vypracovania: Ústav informatiky a matematiky

Názov práce: **QC-MDPC McEliece na mikroprocesore**

Špecifikácia zadania:

QC-MDPC kódy sú výhodnou alternatívou pre realizáciu postkvantového McElieceovho kryptosystému vzhľadom na nižšie softvérové nároky. Úlohou je otestovať naportovať a otestovať implementáciu QC-MDPC systému na mikroprocesor.

Úlohy:


1. Naštudujte problematiku.
2. Naportujte riešenie na mikroprocesor.
3. Otestujte riešenie

Zoznam odbornej literatúry:

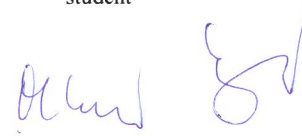
1. A. Gulyás. Implementácia QC-MDPC McElieceovho kryptosystému. Diplomová práca, STU, 2015.
2. Misoczki, Rafael, et al. "MDPC-McEliece: New McEliece variants from moderate density parity-check codes." Information Theory Proceedings (ISIT), 2013 IEEE International Symposium on. IEEE, 2013.
3. Ruhr Universität Bochum. Code-based Cryptography. <http://sha.rub.de/research/projects/code/>


Riešenie zadania práce od: 21. 09. 2015

Dátum odovzdania práce: 20. 05. 2016


Radovan Bezák
študent




prof. RNDr. Otokar Grošek, PhD.
vedúci pracoviska


prof. RNDr. Gabriel Juhás, PhD.
garant študijného programu

SÚHRN

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Študijný program:	Aplikovaná informatika
Autor:	Radovan Bezák
Bakalárska práca:	QC-MDPC McEliece na mikroprocesore
Vedúci záverečnej práce:	doc. Ing. Pavol Zajac, PhD.
Miesto a rok predloženia práce:	Bratislava 2016

Táto bakalárska práca sa zaoberá návrhom a implementáciou asymetrického šifrovacieho systému, založeného na McElieceovom kryptosystéme s využitím QC-MDPC kódov. Tento typ kódovania zabezpečuje zníženie veľkosti kľúčov pri súčasnom zvýšení úrovne bezpečnosti. Výsledkom práce je návrh a prvá testovacia verzia vzorovej implementácie šifrovacieho systému μ Eliece, ktorý ponúka 128-bitovú bezpečnosť a je dostatočne nenáročný pre beh na zariadeniach s nízkou výpočtovou silou. V rámci práce implementujeme testovaciu verziu μ Eliece pre OS Linux a pre 8-bitový mikropočítač Atmega1284P. Základ v teórii kódovania robí z kryptosystému μ Eliece kandidáta na post-quantovú kryptografiu.

Kľúčové slová: McEliece, kódovanie, asymetrická, postkvantová, PQ, kryptografia, mikroprocesor

ABSTRACT

SLOVAK UNIVERSITY OF TECHNOLOGY IN BRATISLAVA

FACULTY OF ELECTRICAL ENGINEERING AND INFORMATION TECHNOLOGY

Study Programme:	Applied Informatics
Author:	Radovan Bezák
Bachelor Thesis:	QC-MDPC McEliece on a microprocessor
Supervisor:	doc. Ing. Pavol Zajac, PhD.
Place and year of submission:	Bratislava 2016

This bachelor's thesis deals with the design and implementation of a public key cryptosystem, based on the McEliece cryptosystem using QC-MDPC codes. This type of code lets us lower the key size while raising the level of security. The result of this project is the design and the first, proof-of-concept version of the sample implementation of a cryptosystem called μ Eliece. This cryptosystem offers us 128-bit security and is lightweight enough to run on low-power devices. Within this work we implement the proof-of-concept version of μ Eliece for OS Linux and for an 8-bit microcontroller Atmega1284P. Being based on coding theory makes the μ Eliece cryptosystem a post-quantum cryptography candidate.

Keywords: McEliece, code-based, asymmetric, post-quantum, PQ, cryptography, micro-processor

Pod'akovanie

Týmto by som chcel vyjadriť pod'akovanie vedúcemu práce, doc. Ing. Pavlovi Zajacovi, PhD., za ponuku zaujímavej témy, ochotu a venovaný čas. Taktiež chcem pod'akovať aj ďalším pedagógom našej fakulty, ktorým skutočne ide o študentov a ich práca je pre nich povolaním, nie len zamestnaním.

Obsah

Úvod	11
1 Analýza problému	12
1.1 McElieceov kryptosystém	12
1.2 QC-MDPC McEliece	12
1.2.1 Parametre QC-MDPC McEliece	12
1.2.2 Algoritmy QC-MDPC McEliece	13
1.2.3 Dekódovanie QC-MDPC kódu	14
1.3 CCA-2 zabezpečenie	15
1.4 Cieľová platforma a jej obmedzenia	15
2 Návrh kryptosystému	17
2.1 QC-MDPC MECS	17
2.2 CCA-2 konverzia	18
2.3 Štruktúra údajov	19
2.4 Postup šifrovania	21
2.5 Postup dešifrovania	22
3 Implementácia	23
3.1 QC-MDPC MECS	23
3.2 CCA-2 konverzia	24
3.3 Rozhranie	24
3.4 Generátor kľúčového páru	25
3.5 Testovacie programy	26
3.5.1 OS GNU/Linux: uEl-file	26
3.5.2 AVR Atmega1284P: EncDec	27
4 Výsledky	28
4.1 Výsledky prvých testov	28
4.2 Možnosti vývoja v ďalších verziách	29
4.2.1 Odlišnosti od špecifikácie	29
4.2.2 Možnosti optimalizácie	29
Záver	31
Zoznam použitej literatúry	32

Prílohy	I
A Manuál k priloženým programom	II
A.1 knižnica uEliece (Programs/*/uEliece/)	II
A.2 x86/uEl-file	II
A.3 x86/uEl-keygen	II
A.4 AVR/Atmega1284P/EncDec	III
B Štruktúra elektronického nosiča	V

Zoznam obrázkov a tabuliek

Obrázok 1	Schéma jadra AVR[1]	16
Obrázok 2	Schéma CCA-2 konverzie μ Eliece	18
Obrázok 3	Schéma CCA-2 konverzie γ podľa [5]. Zdroj obrázka: [13]	19
Obrázok 4	Schéma použitia testovacej implementácie μ Eliece na šifrovanie . .	25
Obrázok 5	Schéma použitia testovacej implementácie μ Eliece na dešifrovanie .	25
Tabuľka 1	Odporúčané parametre QC-MDPC MECS podľa [7]	13
Tabuľka 2	μ Eliece_0.7_alpha - čas šifrovania/dešifrovania	28
Tabuľka 3	μ Eliece_0.7_alpha - čas CCA2 konverzie	28
Tabuľka 4	μ Eliece_0.7_alpha - čas šifrovania/dešifrovania MECS	28

Zoznam skratiek a značiek

MECS - McElieceov kryptosystém[6]

QC - kvázicyklické

MDPC - Moderate-density parity-check kódovanie[10]

m - nešifrovaná správa - plaintext

c - šifrovaná správa - ciphertext

Zoznam algoritmov

1	Pseudokód Gallagerovho bit-flipping algoritmu[3]	14
---	--	----

Úvod

Téma post-quantovej kryptografie sa v poslednom čase stáva stále viac diskutovanou a aktuálnou. Americký NIST (*National Institute of Standards and Technology*) na konferencii PQ Crypto 2016 oznámil, že pripravuje štandardizačnú súťaž pre post-quantové kryptografické algoritmy a ešte v roku 2016 zverejní výzvu na zaslanie príspevkov[9].

Jednou z vhodných možností post-quantovej kryptografie sú šifrovacie algoritmy založené na teórii kódovania. Tieto oproti väčšine ďalších adeptov vynikajú rýchlosťou a bezpečnosťou. Najznámejšie kryptografické schémy, založené na teórii kódovania, sú McElieceova a Niederreiterova schéma. V čase publikácie ich najmä veľká dĺžka kľúčov robila nepraktickými. Vďaka novej pozornosti však napredoval vývoj a boli navrhnuté nové varianty týchto schém, ktoré majú praktickejšie vlastnosti.

V tejto práci navrhujeme a implementujeme šifrovací systém, založený na jednej z nich - McEliece s využitím kvázicyklických MDPC kódov. Navrhovaný šifrovací systém poskytuje 128-bitovú úroveň bezpečnosti, je ľahko prenosný a dostatočne nenáročný, aby mohol byť použitý na širokom spektre zariadení. Toto prezentujeme implementáciou navrhovaného kryptosystému pre operačné systémy GNU/Linux a taktiež pre 8-bitové mikropočítače AVR Atmega1284P.

V prvej kapitole stručne popisujeme tematiku QC-MDPC McEliece kryptosystému, nevyhnutného CCA-2 zabezpečenia a tiež cieľovú platformu mikropočítača Atmega. V nasledujúcej kapitole popisujeme navrhovaný šifrovací systém, jeho jednotlivé komponenty, algoritmy a štruktúru výslednej správy. Na základe tejto kapitoly by mal byť čitateľ schopný vytvoriť vlastnú kompatibilnú implementáciu kryptosystému. Tretia kapitola je venovaná opisu našej implementácie a prístupov, ktoré sme si pre ňu zvolili. V poslednej kapitole sú uvedené výsledky testovania prvej verzie našej implementácie navrhovaného kryptosystému a možnosti jej ďalšieho vývoja.

1 Analýza problému

1.1 McElieceov kryptosystém

McElieceov kryptosystém[6] (ďalej MECS) je asymetrický šifrovací algoritmus založený na teórii kódovania. Návrh tohto algoritmu bol publikovaný profesorom Robertom J. McEliece-om v roku 1978. Je prvým asymetrickým kryptosystémom, využívajúcim znáhodnenie. Podstatou McElieceovho kryptosystému je zakódovať správu do kódového slova pomocou kódu, schopného opraviť istý počet chýb. Tieto chyby sú následne náhodne do kódového slova vnesené. Prijímateľ správy je schopný pomocou súkromného kľúča - kontrolnej matice kódu - chyby odstrániť a kódové slovo dekódovať, čím získa nešifrovanú správu.

Pôvodný McElieceov kryptosystém využíva Goppa kódy, ktoré sa vyznačujú veľkou veľkosťou kľúčov. Toto robí McElieceov kryptosystém veľmi nepraktickým oproti dnes bežne používaným asymetrickým šifrovacím algoritmom, založeným na teórii čísel. Kľúče týchto algoritmov sú pri rovnakej úrovni bezpečnosti niekoľkonásobne menšie. Aj z tohto dôvodu McElieceov kryptosystém nenašiel praktické uplatnenie.

1.2 QC-MDPC McEliece

McElieceov kryptosystém sa opäť stal zaujímavou témou, keď sa začalo viac uvažovať o možnostiach konštrukcie kvantového počítača a o post-quantovej kryptografii. Zatiaľ čo šifrovacie systémy, založené na teórii čísel, sú efektívne prelomiteľné pomocou kvantových algoritmov[11], dekódovanie všeobecného lineárneho kódu je známy NP-úplný problém a efektívny algoritmus na jeho riešenie zatiaľ neexistuje.

Postupne vznikali nové varianty McElieceovho kryptosystému, využívajúce nové typy kódovania, ktoré umožňovali vyššiu úroveň bezpečnosti a nižšiu veľkosť kľúčov. Jedným z týchto nových typov kódov sú kvázicyklické (ďalej QC) MDPC kódy[10]. Na ich základe bol navrhnutý QC-MDPC McElieceov kryptosystém[7]. Verejný aj súkromný kľúč tohto kryptosystému tvoria kvázicyklické matice, čo znamená, že ich môžeme reprezentovať prvým riadkom a tým pádom veľmi výrazne znížiť veľkosť kľúčov.

1.2.1 Parametre QC-MDPC McEliece

Predtým, než opíšeme jednotlivé algoritmy QC-MDPC MECS, uvádzame ich parametre. Kontrolná matica (označujeme H) QC-MDPC kódu je $n \times k$ QC matica, zložená z n_0 horizontálne uložených $k \times k$ cyklických blokov, ktoré označujeme $H_0 \dots H_{n_0-1}$. Každý riadok kontrolnej matice má rovnomerne rozloženú váhu w . Generujúca matica G je $k \times (n - k)$ QC matica zložená z $n_0 - 1$ vertikálne usporiadaných $k \times k$ cyklických blokov,

pripojená sprava na jednotkovú maticu I . Pri dekódovaní takéhoto kódu musíme byť schopní opraviť t chýb.

Pre účely QC-MDPC McEliece kryptosystému autori odporúčajú určité nastavenia týchto parametrov[7]. Tieto uvádzame v nasledujúcej tabuľke 1.

Tabuľka 1: Odporúčané parametre QC-MDPC MECS podľa [7]

Úroveň bezpečnosti	n_0	n	k	w	t
80	2	9602	4801	90	84
80	3	10779	3593	153	53
80	4	12316	3079	220	42
128	2	19714	9857	142	134
128	3	22299	7433	243	85
128	4	27212	6803	340	68
256	2	65542	32771	274	264
256	3	67593	22531	465	167
256	4	81932	20483	644	137

1.2.2 Algoritmy QC-MDPC McEliece

QC-MDPC McElieceov kryptosystém pozostáva z nasledujúcich algoritmov:

- Generovanie kľúčového páru:

1. Vygenerujeme $n_0 - 1$ náhodných cyklických matíc $H_0 \dots H_{n_0-2}$ dĺžky k s váhou w

2. Vygenerujeme náhodnú cyklickú invertibilnú maticu H_{n_0-1} dĺžky k s váhou w

3. $H = \begin{bmatrix} H_0 & H_1 & \dots & H_{n_0-1} \end{bmatrix}$ je súkromný kľúč.

4. $Q = \begin{bmatrix} (H_{n_0-1}^{-1} \cdot H_0)^T \\ (H_{n_0-1}^{-1} \cdot H_1)^T \\ \dots \\ (H_{n_0-1}^{-1} \cdot H_{n_0-2})^T \end{bmatrix}$

5. $G = \begin{bmatrix} I & Q \end{bmatrix}$ je verejný kľúč.

- Šifrovanie:

1. Vygenerujeme náhodný vektor e dĺžky n a váhy t

2. $c = m \cdot G + e$.

- Dešifrovanie:

1. Dekódovacím algoritmom so znalosťou H a c vypočítame chybový vektor e
2. $m = \text{trunc}(c, k) - \text{trunc}(e, k)$

1.2.3 Dekódovanie QC-MDPC kódu

Na dekódovanie QC-MDPC kódov môžeme použiť rovnaké algoritmy ako na dekódovanie ich predchodcov, LDPC kódov[3]. Jedným z nich je Gallagerov Bit flipping algoritmus a jeho rôzne varianty[3, 4]. Tento algoritmus najprv vypočíta syndróm prijatej správy $s = c \cdot H^T$. V následných iteráciách na základe syndrómu pre každý bit vypočíta počet nesplnených kontrol parity ($\#upc$). Pokiaľ $\#upc$ prekročí stanovenú hranicu, bit je preklopený a aktualizuje sa syndróm. Algoritmus pokračuje, až kým sa nedosiahne syndróm $s = 0$, čo znamená, že boli odhalené všetky chyby. V prípade dosiahnutia maximálneho počtu iterácií pred dosiahnutím $s = 0$ sa dekódovanie preruší a prehlási sa za neúspešné. Hranica $\#upc$ pre preklopenie bitu môže byť stanovená pevne na vopred vypočítané optimálne hodnoty (pre jednotlivé iterácie algoritmu) alebo počítaná dynamicky v priebehu algoritmu. Menením tejto vlastnosti algoritmu môžeme dosahovať rozličné výsledky vzhľadom na rýchlosť a úspešnosť dekódovania.

Algoritmus 1 Pseudokód Gallagerovho bit-flipping algoritmu[3]

```

1  vypocitaj syndrom;
2  for i: 0 -> i_max do
3      for j: 0 -> N do
4          vypocitaj #upc pre c[j];
5          if ( #upc > limit[i] ) then
6              flip c[j];
7          end if
8      end for
9      aktualizuj syndrom;
10     if ( syndrom is zero ) then
11         return DECODING_SUCCESSFUL;
12     end if
13 end for
14 return DECODING_FAILURE;
```

1.3 CCA-2 zabezpečenie

Z algoritmu pre šifrovanie QC-MDPC MECS vyplýva, že $\text{trunc}(c, k) = \text{trunc}(m, k) + \text{trunc}(e, k)$. Znamená to, že šifrovaný text sa od nešifrovaného líši maximálne v t bitoch. Nakolko pomer počtu chýb t a dĺžky správy k je pri odporúčaných parametroch veľmi malý, znamená to, že šifrovaná správa môže byť stále z väčšej časti čitateľná a zneužitelná. Navyše ak sa útočníkovi podarí viac krát zachytiť šifrovanú rovnakú správu (napríklad v prípade neúspešného dekódovania a opätovného zaslania správy), otvára mu to možnosti rôznych útokov.[13] Sčítaním dvoch šifrovaných správ dokáže zistiť $e_1 + e_2 = c_1 + c_2$.

Tieto problémy môžeme vyriešiť pridaním CCA-2 bezpečnostnej obálky pred samotným šifrovaním McEliece kryptosystémom. CCA-2 zabezpečenie vratne zamaskuje správu tak, že je nerozoznateľná od náhodného vektora. Tiež zabezpečí rozdielnosť výsledných zašifrovaných správ aj v prípade, ak ide o opakované šifrovanie rovnakého nešifrovaného textu. V práci vychádzame z návrhov schém CCA-2 konverzie McElieceovho kryptosystému, ktoré publikovali Kobara a Imai[5].

1.4 Cieľová platforma a jej obmedzenia

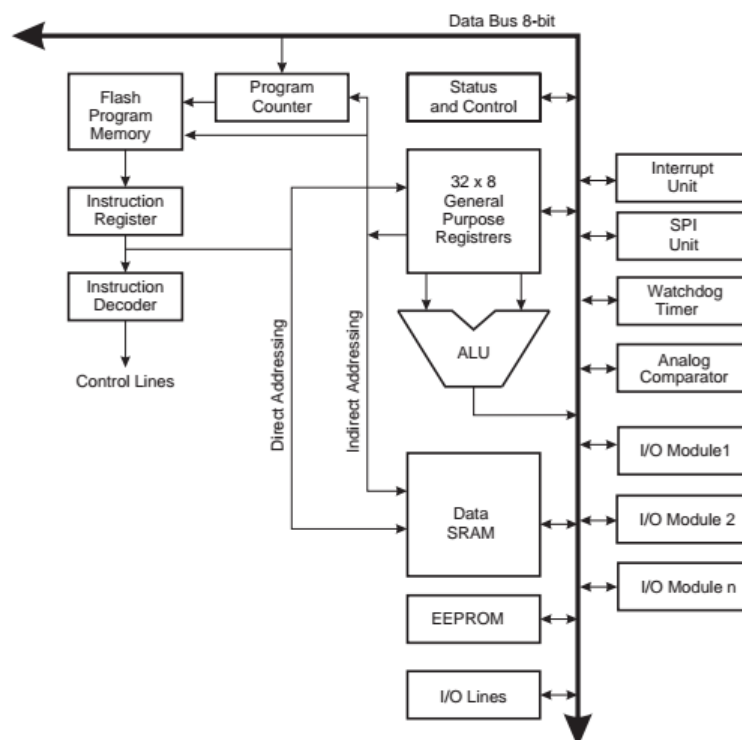
Informácie v tejto kapitole sú čerpané z oficiálnej dokumentácie mikropočítača ATmega1284P.[1]

Cieľovou platformou tohto projektu sú 8-bitové mikrokontrolery zo série megaAVR. Zariadenia tejto série využívajú architektúru jadra AVR (Obr. 1). Architektúra AVR využíva Harvardskú architektúru počítača, so samostatnými pamäťovými jednotkami a zbernicami pre program a pre údaje. Počas vykonávania inštrukcie sa už z programovej pamäte načítava nasledujúca inštrukcia, čím sa umožňuje vykonávanie inštrukcií v každom jednom cykle procesora. Procesor využíva rozšírený RISC súbor inštrukcií.

Mikrokontrolery série megaAVR sú jednočipové počítače so širokým rozsahom parametrov a množstvom periférnych funkcií. Procesor dokáže spracovať 131 inštrukcií.

Pre vyhovujúce parametre počítača sme si pre účely tohto projektu vybrali model ATmega1284P. Tento model dokáže pracovať s frekvenciou maximálne 20MHz. Má 32 vstupno/výstupných programovateľných pinov. Ponúka 128kB vnútorne programovateľnej programovej flash pamäte, 4kB pamäte EEPROM a 16kB operačnej pamäte SRAM. K ďalším funkciám patria dve sériové rozhrania USART, ktoré môžeme použiť v rámci projektu na testovacie účely. Model ATmega1284P má aj množstvo ďalších funkcií, ktoré síce v rámci projektu využívať nebudeme, no je užitočné ich spomenúť, keďže najpravdepodobnejším využitím výsledkov tohto projektu v praxi je konštrukcia rôznych senzorov a zariadení s pripojením k internetu alebo inej sieti, ktoré dokážu v šifrovanej forme zabezpečiť

Obrázok 1: Schéma jadra AVR[1]



prenášať získané údaje. K týmto funkciám patria napríklad analog-to-digital converter (ADC), ktorý umožňuje pripojenie rôznych meracích sond a zariadení, podpora systémových prerušení alebo dve sériové rozhrania USART, umožňujúce sériovú komunikáciu s inými zariadeniami. Nakoľko zariadenia tohto typu sú často napájané batériou alebo solárnou energiou, dôležitá je aj nízka spotreba energie a viacero možností úsporných režimov.

Pre náš projekt sú teda dôležité najmä nasledujúce parametre, funkcie a obmedzenia:

- Frekvencia procesora: 1MHz – 20MHz
- Veľkosť programovej pamäte Flash: 128kB
- Veľkosť pamäte EEPROM: 4kB
- Veľkosť operačnej pamäte SRAM: 16kB
- sériové rozhranie USART na testovanie riešenia

2 Návrh kryptosystému

Jedným z výstupov tejto práce je návrh šifrovacieho systému s názvom μ Eliece, založeného na QC-MDPC McEliece kryptosystéme a využívajúceho nami poupravenú schému CCA-2 konverzie γ navrhnutú v článku [5]. Šifrovací systém μ Eliece má poskytovať 128-bitovú bezpečnosť. Taktiež musí byť pomerne ľahko portovateľný a nenáročný, aby bol použiteľný na rôznych platformách - osobné počítače, servery, mikropočítače.

Vďaka zvolenej schéme CCA-2 konverzie sa dá náš kryptosystém klasifikovať ako hybridný - zložený z prúdového (symetrického) a asymetrického šifrovania. Správa ľubovoľnej dĺžky sa šifruje prúdovou šifrou pomocou náhodne generovaného kľúča a len jej posledný blok dĺžky k následne podlieha šifrovaniu MECS, ktorým sa zabráni dešifrovaniu prúdovej šifry. Táto schéma má aj tú výhodu, že výpočtovo náročnejší proces MECS prebieha pre ľubovoľne dlhú správu len jeden krát. Rýchlosť šifrovania je teda so zvyšujúcou sa dĺžkou správy čím ďalej, tým viac závislá len od rýchlosti prúdového šifrovania (CCA-2 konverzie.)

2.1 QC-MDPC MECS

Pre potreby tohto šifrovacieho systému sme spomedzi odporúčaných súborov parametrov pre QC-MDPC MECS[7] zvolili nasledovný:

- úroveň bezpečnosti = 128bit
- $n_0 = 2$
- $n = 19714$
- $k = 9857$
- $w = 142$
- $t = 134$

Tieto parametre boli vybrané kvôli tomu, že 128-bitová úroveň bezpečnosti bola jednou zo stanovených požiadaviek, poskytovala najnižšiu minimálnu dĺžku šifrovanej správy a najnižšiu veľkosť kľúčov. Následkom voľby parametra $n_0 = 2$ tvorí verejný kľúč kvázicyklická matica tvorená len jedným cyklickým blokom, čiže cyklická matica. Tým sa zjednodušuje postup rotácie a teda sa znižuje zložitosť kódovania.

2.2 CCA-2 konverzia

Pri návrhu CCA-2 konverzie kryptosystému μ Eliece sme vychádzali zo schémy γ (na Obr. 3) z článku [5]. Táto schéma bola autormi článku [13] overovaná a označená za bezpečnú.

Obrázok 2: Schéma CCA-2 konverzie μ Eliece

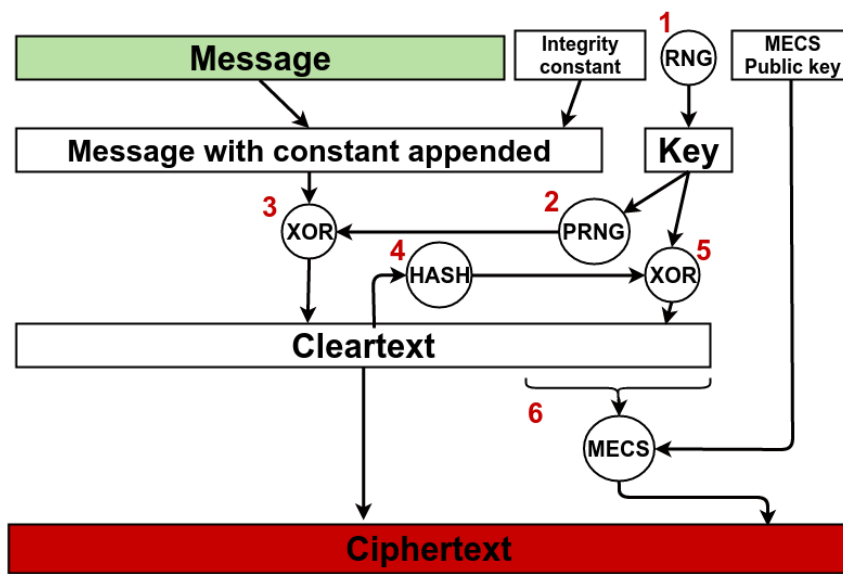
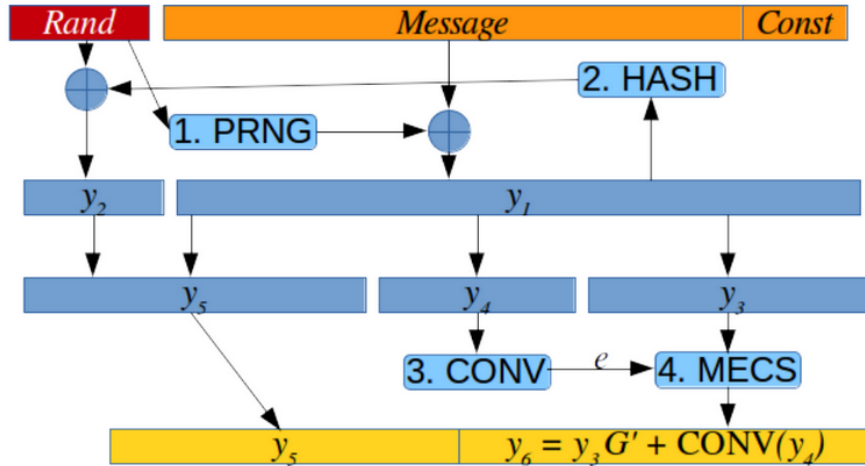


Schéma bola pre použitie v μ Eliece modifikovaná najmä pre jej zjednodušenie a tiež pre prispôbenie možnosti budúcej implementácie šifrovania 'on-the-fly'. Z pôvodnej schémy bola vynechaná funkcia CONV, ktorá pretvorila časť šifrovanej správy na chybový vektor, ktorý bol použitý pri šifrovaní MECS. Týmto sa skrátila celková dĺžka odosielanej správy. Túto časť sme v μ Eliece vynechali kvôli zložitosti samotnej funkcie CONV, súvisiacemu zvyšovaniu zložitosti kódu a zvýšeným požiadavkám na zdroje pri dešifrovaní správy. Nie je nám známe, že by aplikácia funkcie CONV v pôvodnej schéme γ mala bezpečnostný význam.

Ďalšou modifikáciou oproti pôvodnej schéme je presunutie zašifrovaného náhodne generovaného kľúča prúdovej šifry zo začiatku správy na jej koniec. Táto časť musí byť pred odoslaním zašifrovaná hashom celého zvyšku správy. Jej umiestnenie na začiatok správy preto znamená, že žiadna časť správy nemôže byť odoslaná a odstránená z pamäti zariadenia pred dokončením celej konverzie. Toto môže byť nevýhoda pri zariadeniach s obmedzenými zdrojmi alebo pri šifrovaní veľkých objemov dát. Po presunutí tejto časti na koniec konvertovanej správy bude možné vytvoriť implementácie kryptosystému, ktoré ihneď po zašifrovaní časti správy prúdovou šifrou a jej následnej konzumácii hashovacou funkciou ju môžu odoslať alebo zapísať na disk a odstrániť z pamäti zariadenia. V pamäti

zariadenia bude musieť byť uložená len časť správy dĺžky potrebnej na šifrovanie MECS. Týmto zmenami sme dospeli k výslednej schéme použitej CCA-2 konverzie, ktorá je znázornená na Obrázku 2.

Obrázok 3: Schéma CCA-2 konverzie γ podľa [5]. Zdroj obrázka: [13]



Predvolená hodnota konštanty na overenie integrity správy (ICK) je 32 byte s hodnotou `0x80`. Túto hodnotu si môžu užívatelia kryptosystému určiť ľubovoľne, je však dôležité, aby komunikujúce strany mali nastavenú rovnakú hodnotu, inak bude správa pri dešifrovaní vždy označená za pozmenenú a nebude možné odhaliť skutočne falšovanú správu. Vo väčšine prípadov použitia je najrozumnejšie ponechať predvolenú hodnotu.

2.3 Štruktúra údajov

Kryptosystém μ Eliece pracuje na 8-bitových slovách. Za prvý bit, teda bit s indexom 0 považujeme *least-significant bit* prvého 8-bitového slova (byte s indexom 0.) Za nasledujúci bit považujeme ten, ktorý nasleduje v smere zvyšujúcej sa významnosti bitov. Keď sa dostaneme na *most-significant bit* 8-bitového slova, za nasledujúci bit sa považuje *least-significant bit* nasledujúceho slova. V prípade nasledovnej reprezentácie správy v jazyku C: `uint8_t message[]`; môžeme teda vyjadriť hodnotu konkrétneho bitu s indexom i týmto výrazom: $((\text{message}[i / 8] \gg (i \% 8)) \& 1)$

Šifrovaná správa sa skladá z nasledujúcich častí (v poradí):

1. `msg_len` byte správa
2. 1 byte padding

3. 32 byte konštanta na overenie integrity (ICK)
4. $(\text{msg_len} < 1167) ? 1167 - \text{msg_len} : 0$ byte padding
5. 32 byte kľúč prúdovej šifry (**sskey**)
6. 1 bit padding
7. 7 bitov technický padding*
8. 1232 byte + 1 bit paritné bity
9. 7 bitov technický padding*

* - *technický padding* znamená, že sa pri operáciách súvisiacich so šifrovaním MECS neberie do úvahy. Slúži len na zarovnanie na 8-bitové slová, kvôli zjednodušeniu manipulácie.

Prvých 9857 bitov posledných 1233 byte šifrovanej správy nazývame *paritný blok*. Prvých 9857 bitov predchádzajúcich 1233 byte nazývame *kódovaný blok*. Časť správy od začiatku až po **sskey** nazývame *konvertovaný blok*. Z konštrukcie kryptosystému vyplýva, že dĺžka zašifrovanej správy je:

$$((\text{msg_len} < 1167) ? 1167 : \text{msg_len}) + 66 + 1233 \text{ byte}$$

To znamená 2466 byte pre plaintext dĺžky nižšej alebo rovnej 1167 byte alebo $(\text{msg_len} + 1299)$ byte pre dlhší plaintext.

Kľúčový pár odporúčame reprezentovať nasledovne:

- **verejný kľúč (generujúca matica G):**

Prvý riadok pravej časti matice G ako vektor bitov, uložených v 8-bitových slovách v zmysle indexovania bitov popísaného vyššie. Tj. 1233 byte, pričom posledných 7 bitov pri šifrovaní neberieme do úvahy.

- **súkromný kľúč (kontrolná matica H):**

Prvý riadok matice H ako vektor $t = 142$ 16-bitových čísel, označujúcich indexy nenulových bitov v riadku. Spolu 284 byte.

2.4 Postup šifrovania

Šifrovanie správy prebieha v nasledovných krokoch:

1. Za plaintext správu pripojíme 1 padding byte s hodnotou 0x80.
2. Pripojíme 32 byte ICK.
3. Pripojíme $((\text{msg_len} < 1167) ? 1167 : \text{msg_len})$ padding byte s hodnotou 0x00.
4. Získali sme *konvertovaný blok*.
5. Vygenerujeme náhodný 32 byte **sskey**.
6. **sskey** nastavíme ako vstup hashovacej funkcie SHAKE128.
7. Získame výstup SHAKE128 dĺžky *konvertovaného bloku*.
8. *konvertovaný blok* XOR zašifrujeme výstupom SHAKE128.
9. Zašifrovaný *konvertovaný blok* nastavíme ako vstup SHAKE128.
10. Získame výstup SHAKE128 dĺžky 32 byte a XOR zašifrujeme **sskey**.
11. Zašifrovaný **sskey** pripojíme na koniec správy.
12. Pripojíme 1 padding byte s hodnotou 0x00. Posledných 7 bitov tohto byte je technických.
13. Prvých $k = 9857$ bitov z posledných 1233 byte správy (*kódovaný blok*) zakódujeme pomocou verejného kľúča.
14. Získaných $k = 9857$ paritných bitov pripojíme na koniec správy.
15. Pripojíme 7 technických padding bitov s hodnotou **false** (doplnok do celého byte).
16. Vygenerujeme náhodný chybový vektor dĺžky $n = 19714$ a váhy $t = 135$.
17. Chybový vektor XOR aplikujeme na *kódovaný blok* a *paritný blok*. Technické padding bity neberieme pri indexovaní do úvahy.
18. Správa je zašifrovaná a pripravená na odoslanie.

2.5 Postup dešifrovania

Dešifrovanie správy prebieha opačne ako šifrovanie, v nasledovných krokoch:

1. Pomocou znalosti súkromného kľúča dekodujeme *kódovaný blok* a *paritný blok* zašifrovanej správy. Technické padding bity neberieme do úvahy. Pri dekodovaní odstránime chyby.
2. Nastavíme *konvertovaný blok* na vstup SHAKE128.
3. Získame výstup SHAKE128 dĺžky 32 byte a XOR dešifrujeme ním zašifrovaný **sskey**.
4. Nastavíme dešifrovaný **sskey** na vstup SHAKE128.
5. Získame výstup SHAKE128 dĺžky *konvertovaného bloku* a XOR dešifrujeme ním zašifrovaný *konvertovaný blok*.
6. Nájdeme ICK - posledných 32 nenulových byte dešifrovaného *konvertovaného bloku*.
7. Porovnáme nájdenu ICK s očakávanou hodnotou pre overenie integrity prijatej správy.
8. Pred ICK sa nachádza byte s hodnotou 0x80 a pred ním je posledný byte dešifrovanej správy.
9. Vyberieme dešifrovanú správu.

3 Implementácia

Výstupom našej práce je aj prvá verzia vzorovej implementácie kryptosystému μ Eliece. Táto implementácia je funkčná na osobných počítačoch s operačným systémom GNU/Linux a na mikropočítači Atmega1284P. Implementácia je integrovaná vo forme knižnice, poskytujúcej jednoduché rozhranie - funkcie na šifrovanie a dešifrovanie správy.

V čase písania je v 'proof-of-concept' verzii 0.7_alpha a je určená na preukázanie funkčnosti a testovanie navrhovaného šifrovacieho systému. CCA-2 konverzia je v tejto verzii zjednodušená použitím iného hashovacieho algoritmu, keďže sme v čase implementácie nemali k dispozícii žiadnu vhodnú implementáciu SHAKE128. Tiež v implementácii pre AVR využívame zjednodušenú, dočasnú náhradu generovania náhodných čísel. Implementácia plnohodnotne preukazuje funkčnosť schémy CCA-2 konverzie a QC-MDPC McElieceovho kryptosystému. Vo verzii 1.0 bude vhodná už aj na bezpečné šifrovanie údajov a kompatibilná so všetkými nasledujúcimi verziami.

3.1 QC-MDPC MECS

Na dekódovanie používame variant Gallagerovho bit-flipping algoritmu, v ktorom máme prednastavené hodnoty prahu preklopenia bitu pre jednotlivé iterácie. V každej iterácii prechádzame všetky bity kódovaného a paritného bloku správy (ktoré spolu tvoria kódové slovo), pre každý vypočítame $\#upc$ tak, že spočítame nenulové bity v prislúchajúcom riadku H^T na pozíciách nenulových bitov syndrómu správy a pokiaľ je výsledná hodnota vyššia než nastavená hodnota prahu pre danú iteráciu, bit je preklopený (zmení sa jeho hodnota na opačnú). Po preklopení bitu sa aktualizuje syndróm správy odčítaním riadka kontrolnej matice, prislúchajúceho danému bitu. Po aktualizácii syndrómu sa skontroluje, či nie je nulový. Ak je syndróm nulový, dekódovací algoritmus sa úspešne ukončí.

Prahové hodnoty pre jednotlivé iterácie algoritmu sme v našej implementácii experimentálne určili na $\{43, 41, 41, 40, 40, 38, 40, 46, 46, 44\}$. Maximálny počet iterácií sme nastavili na 10. Pri testovaní týchto parametrov sme dosiahli výsledky:

- priemerný počet iterácií algoritmu 3.06936
- priemerný počet preklopení bitov 141.2204
- početnosť zlyhaní dekódovania 0.03%.

Výsledky testov sú pre tieto hodnoty veľmi priaznivé a vidíme už len malý priestor na

zlepšenie prednastavených prahových hodnôt. Varianta algoritmu s prednastavenými prahovými hodnotami bola vybratá kvôli nižším nárokom na výpočtové zdroje a jednoduchšej implementácii.

3.2 CCA-2 konverzia

Šifrovanie konvertovaného bloku správy prúdovou šifrou prebieha po 256-bitových častiach. To znamená, že pre každú časť získame 256-bitový výstup z hash funkcie, zašifrujeme ním časť a prejdeme na ďalšiu. Vo verzii `0.7_alpha` našej implementácie nevyužívame hash algoritmus `SHAKE128` s výstupom variabilnej dĺžky, ale algoritmus `SHA3-256` s 256-bitovým výstupom, ktorý využívame tým spôsobom, že pri každom volaní funkcie zvyšujeme vstup o 1. Týmto sa verzia `0.7_alpha` mierne odlišuje od špecifikácie μ Eliece. Nahradením hashovacej funkcie pôvodne navrhovanou dosiahneme zvýšenie rýchlosti aj bezpečnosti šifrovania a dešifrovania.

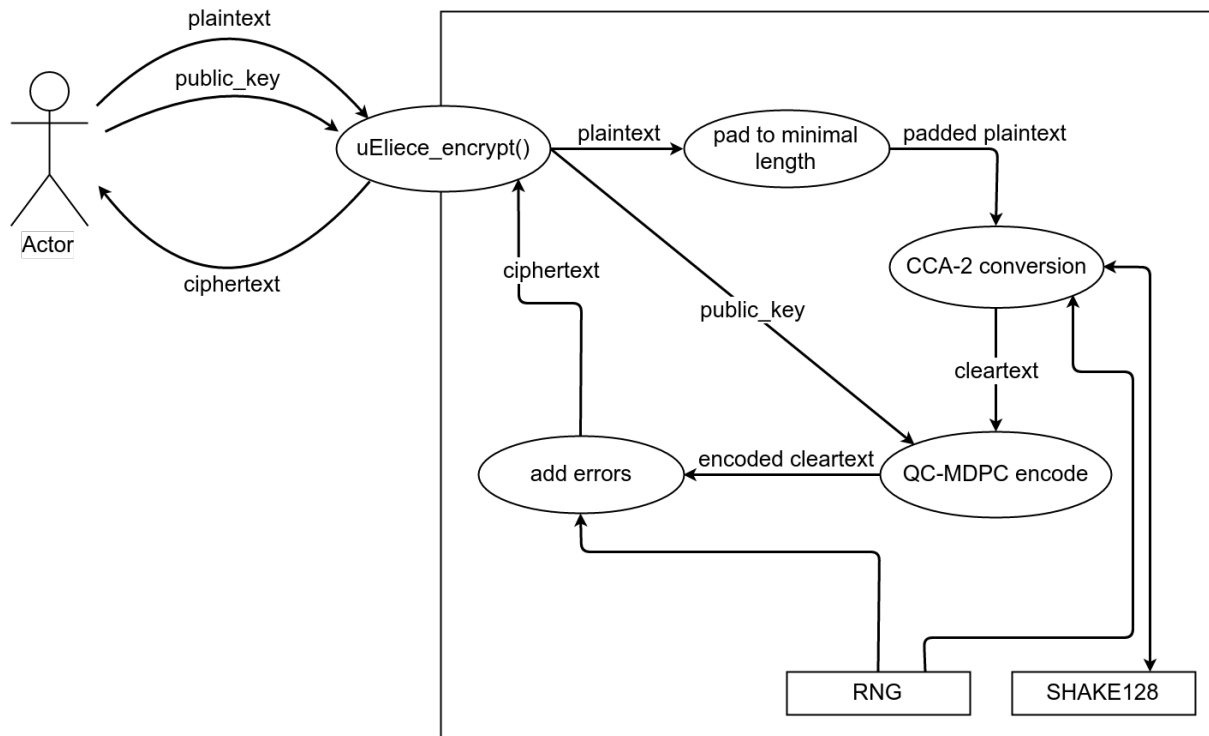
3.3 Rozhranie

Rozhranie implementácie je definované v hlavičkovom súbore `uEliece.h`. Ten deklaruje funkcie, definuje preprocesorové makrá a konštanty prístupné zvonka. Meniť isté nekritické parametre implementácie, ktoré neovplyvňujú súlad so špecifikáciou, môže užívateľ pozmenením hlavičkového súboru `uEliece-settings.h`. Prostredníctvom tohto súboru je možné meniť napríklad parametre dekódovania QC-MDPC kódu alebo hodnotu konštanty `ICK`. Tento hlavičkový súbor obsahuje definície, ktoré sú využívané len zvnútra implementácie kryptosystému, nie je preto potrebné ho vkladať do zdrojového kódu aplikácie. Vložiť do zdrojového kódu je potrebné iba súbor `uEliece.h`.

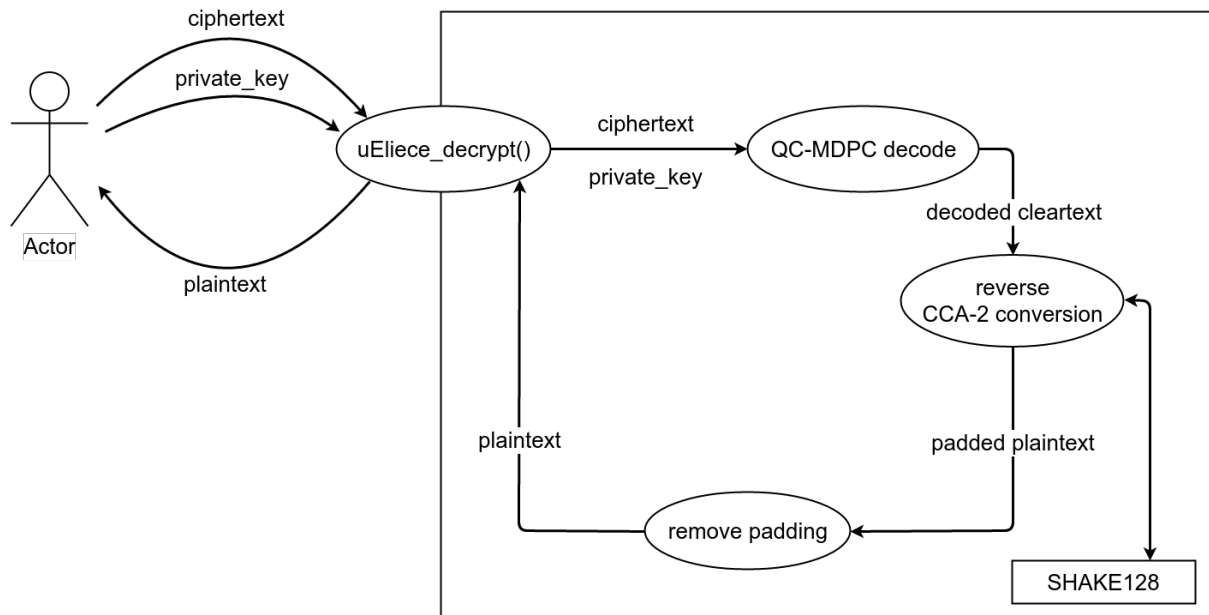
Hlavičkový súbor `uEliece.h` obsahuje napríklad tieto najdôležitejšie definície:

- deklarácie funkcií `uEliece_encrypt()` a `uEliece_decrypt()` a základný popis ich použitia
- definície dátových typov pre súkromný a verejný kľúč
- definície makier na zistenie chýb z návratovej hodnoty funkcií

Obrázok 4: Schéma použitia testovacej implementácie μ Eliece na šifrovanie



Obrázok 5: Schéma použitia testovacej implementácie μ Eliece na dešifrovanie



3.4 Generátor kľúčového páru

Generátor kľúčového páru nie je súčasťou súčasnej verzie našej implementácie μ Eliece. Namiesto toho bol vytvorený ako samostatná aplikácia v jazyku C++ s pomocou knižnice

NTL[12]. Program `uEl-keygen` je konzolová aplikácia, ktorá prijíma jeden argument: `-C`, `-H`, `-K` alebo `-B`. Tento argument určuje formát výstupných súborov, obsahujúcich kľúčový pár.

- `-C` : Výstup do ASCII-kódovaného textového súboru, formátovaného ako inicializátor pola čísel (verejný kľúč: `uint8_t`, súkromný kľúč: `uint16_t`) v programovacom jazyku C. Takto generovaný kľúčový pár môže byť zahrnutý do zdrojového kódu programu.
- `-H` : Výstup do ASCII-kódovaného textového súboru, formátovaného ako hexadecimálne číslo.
- `-K` : Výstup do ASCII-kódovaného textového súboru, formátovaného ako poradie hexadecimálnych čísel (8-bit, 16-bit), oddelených znakom `:`.
- `-B` : Výstup do binárneho súboru.

Argument `-?` zobrazí nápovedu. Program s využitím generátora náhodnosti operačného systému vygeneruje náhodný súkromný kľúč, dopočíta k nemu prislúchajúci verejný kľúč a kľúče zapíše do súborov `uEl_priv.key` a `uEl_pub.key`.

3.5 Testovacie programy

Na predvedenie funkčnosti implementovanej knižnice μ Eliece sme vytvorili dva testovacie programy.

3.5.1 OS GNU/Linux: `uEl-file`

`uEl-file` je konzolová aplikácia, ktorá prijíma 3 argumenty:

- Prvý argument určuje, či chceme šifrovať alebo dešifrovať. Musí mať hodnotu `"-e"` pre šifrovanie alebo `"-d"` pre dešifrovanie.
- Druhý argument je názov vstupného súboru.
- Tretí argument je názov výstupného súboru.

Program načíta zadaný vstupný súbor, podľa nastaveného módu ho zašifruje alebo dešifruje a výsledok zapíše do výstupného súboru. Program na šifrovanie a dešifrovanie využíva kľúčový pár, vložený do jeho zdrojového kódu prostredníctvom súboru `uEl_keys.h`. Na zmenu kľúčového páru je teda potrebné nanovo skompilovať program. Pri zachovaní štruktúry súborov z priloženého nosiča a prechádzajúcej kompilácii generátora kľúčového páru je možné získať nový kľúčový pár a vygenerovať nový súbor `uEl_keys.h` spustením príkazu `make key` v zložke obsahujúcej zdrojový kód programu.

3.5.2 AVR Atmega1284P: EncDec

Program samostatne beží na mikropočítači Atmega1284P a komunikuje prostredníctvom sériového rozhrania UART0 mikropočítača. Pri spustení program nastaví komunikačné rozhranie a následne jeho prostredníctvom odošle reťazec "Device initialized.". Potom vstúpi do nekonečného cyklu, na ktorého začiatku odošle reťazec "Waiting for request...". Potom zariadenie čaká na prijatie správy. Očakávaná správa má nasledovný formát:

- 1 byte s hodnotou 'E', 'D' alebo 'T', ktorý oznamuje zariadeniu, či požadujeme šifrovanie, dešifrovanie alebo testovanie (šifrovanie a následné dešifrovanie)
- 16 bitové celé číslo bez znamienka, vyjadrujúce veľkosť nasledujúcej správy
- správa oznámenej veľkosti, ktorú má zariadenie šifrovať/dešifrovať

Po prijatí takejto požiadavky zariadenie odošle späť detaily požiadavky a začne vykonávať požadované akcie. Po ich vykonaní program odošle výslednú správu a dostane sa znova na začiatok nekonečného cyklu, čiže čaká na ďalšiu požiadavku.

4 Výsledky

4.1 Výsledky prvých testov

Pri testovaní verzie 0.7_alpha našej implementácie μ Eliece na osobnom počítači s procesorom Intel® Core™ i5-3210M CPU @ 2.50GHz sme dosiahli výsledky popísané v tabuľkách 2, 3 a 4.

Tabuľka 2: μ Eliece_0.7_alpha - čas šifrovania/dešifrovania

Size[byte]	Encrypt[ms]	Decrypt[ms]
512	74.002	44.633
1024	75.638	46.288
4096	78.835	48.686
32768	92.255	61.806
262144	194.040	164.108
1048576	543.188	513.999
4194304	1933.794	1910.594

Tabuľka 3: μ Eliece_0.7_alpha - čas CCA2 konverzie

Size[byte]	Enc-CCA2[ms]	Dec-CCA2[ms]
512	0.532	0.517
1024	0.545	0.533
4096	1.849	1.847
32768	14.618	14.645
262144	116.362	117.053
1048576	465.430	467.049
4194304	1856.379	1864.041

Tabuľka 4: μ Eliece_0.7_alpha - čas šifrovania/dešifrovania MECS

Size[byte]	Enc-MECS[ms]	Dec-MECS[ms]
512	73.468	44.113
32768	77.635	47.160
4194304	77.412	46.550

Na výsledkoch vidíme, že rozdiel medzi časom šifrovania a dešifrovania je približne rovnaký, tvorí ho šifrovanie a dešifrovanie QC-MDPC McEliece kryptosystémom. Nakoľko

šifrovanie/dešifrovanie MECS prebieha len jeden krát pre ľubovoľnú veľkosť správy, jeho trvanie je naprieč rôznymi veľkosťami správ takmer konštantné, vid. tabuľka 4. Dekódovanie QC-MDPC kódu pri dešifrovaní trvá kratšie, než zakódovanie pri šifrovaní, kvôli možnosti práce s riedkymi maticami.

Predlžovanie šifrovania v závislosti od dĺžky správy je len vecou CCA2 konverzie, ktorá trvá približne 0.5ms na 1kB šifrovaných dát (tabuľka 3). Rýchlosť CCA2 konverzie je v oboch smeroch takmer rovnaká, keďže ide o veľmi podobný sled operácií.

Pri testovaní na mikropočítači AVR Atmega1284P s taktom procesora 8.00MHz sme namerali dĺžku šifrovania približne 3 minúty a 45 sekúnd a dešifrovania približne 3 minúty a 20 sekúnd. Na testovanie bola použitá správa dĺžky 55 byte.

4.2 Možnosti vývoja v ďalších verziách

V tejto sekcii uvádzame najvýznamnejšie možnosti ďalšieho vývoja súčasnej verzie implementácie μ Eliece, na ktorých budeme ďalej pracovať. Po ich zavedení očakávame výrazné zrýchlenie šifrovania a dešifrovania a taktiež zvýšenie bezpečnosti šifrovacieho systému. Všetky odlišnosti od špecifikácie budú odstránené najneskôr do verzie 1.0, ktorá bude kompatibilná so všetkými nasledujúcimi verziami.

4.2.1 Odlišnosti od špecifikácie

- **Hashovacia funkcia** - ako už sme spomínali vyššie, verzia 0.7_alpha využíva namiesto algoritmu SHAKE128 algoritmus SHA3-256. V najbližších verziách bude algoritmus nahradený špecifikovaným SHAKE128, čím sa zvýši bezpečnosť a rýchlosť šifrovania a dešifrovania. Tiež sa tým zaistí kompatibilita s budúcimi verziami implementácie.
- **Generovanie náhodných čísel v implementácii pre AVR** - nakoľko mikropočítač Atmega1284P nemá zabudovaný žiadny vlastný generátor náhodných čísel alebo automatické získavanie a skladovanie entropie, je potrebné túto funkcionálnosť zahrnúť do implementácie μ Eliece. Vo verzii 0.7_alpha je toto zabezpečené zjednodušene, pomocou generátora pseudonáhodných čísel zo štandardných knižníc jazyka C, ktorého *seed* je pevne nastavený v zdrojovom kóde programu. Po odstránení tejto odlišnosti bude možné implementáciu pre AVR využiť na šifrovanie údajov.

4.2.2 Možnosti optimalizácie

Implementáciu vo verzii 0.7_alpha je možné ďalej matematicky a programovo optimalizovať. Týmto je možné dosiahnuť značné zvýšenie rýchlosti a efektivity šifrovania a dešifrovania.

- **Zefektívnenie rotácie verejného kľúča pri kódovaní** - rotáciu verejného kľúča pri kódovaní je možné implementovať efektívnejšie, v jazyku Assembly, s využitím vhodných inštrukcií. Po oprave tohto nedostatku očakávame podľa hrubého odhadu aspoň štvornásobné zrýchlenie kódovania (výrazná väčšina času šifrovania pri správach $< 100\text{kiB}$) na 8-bitových platformách.
- **Optimalizácia pre jednotlivé platformy** - program je možné optimalizovať pre jednotlivé platformy. Na 64-bitových platformách je možné dosiahnuť zrýchlenie tým, že budeme pracovať s ôsmimi 8-bitovými slovami naraz, čím môžeme pri iteratívnych činnostiach znížiť počet vykonaných inštrukcií procesora aspoň 8-krát. Na 8-bitových platformách môžeme optimalizovať používanie rôznych dátových typov. Operácie nad dátovými typmi so šírkou viac než 8 bitov potrebujú vykonať viac inštrukcií procesora. Takto môžu byť zdrojom spomalenia aj operácie nad počítadlom cyklických činností.

Záver

V priebehu práce sa nám podarilo navrhnuť a implementovať funkčný hybridný šifrovací systém, založený na teórii kódovania - konkrétne na schéme QC-MDPC McEliece. Tento šifrovací systém spĺňa naše požiadavky, ktorými boli 128-bitová bezpečnosť, portovateľnosť a nenáročnosť. Portovateľnosť a nenáročnosť sme dokázali funkčnou implementáciou na 8-bitový mikropočítač Atmega1284P, ktorý má k dispozícii len 16kB operačnej pamäte SRAM.

Po ďalšom vývoji, rozšírení množiny podporovaných platforiem a odladení odolnosti voči útokom postrannými kanálmi môže byť tu navrhnutý šifrovací systém μ Eliece použitý na bezpečné šifrovanie údajov a taktiež byť kandidátom na post-kvantovo bezpečné šifrovanie údajov.

Zoznam použitej literatúry

- [1] ATMEL. ATmega1284P Datasheet. <http://www.atmel.com/images/doc8059.pdf>. [Online; cit. 18.05.2016].
- [2] BERTONI, G., DAEMEN, J., PEETERS, M., AND VAN ASSCHE, G. The Keccak sponge function family. <http://keccak.noekeon.org/>. [Online; cit. 18.05.2016].
- [3] GALLAGER, R. Low-density parity-check codes. *IRE Trans. Inf. Theor.* 8(1) (2009), 21–28.
- [4] GULYÁS, A. Implementácia QC-MDPC McEliecovho kryptosystému. Master's thesis, ÚIM FEI STU, 2015.
- [5] KOBARA, K., AND IMAI, H. Semantically secure McEliece public-key cryptosystems-conversions for McEliece PKC. *Public Key Cryptography* (2001).
- [6] MCELIECE, R. J. A public-key cryptosystem based on algebraic coding theory. *Deep Space Network Progress* 44 (1978), 114–116.
- [7] MISOCZKI, R. E. A. MDPC-McEliece: New McEliece variants from moderate density parity-check codes. In *2013 IEEE International Symposium on Information Theory Proceedings* (2013), IEEE.
- [8] NIST. SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. <http://dx.doi.org/10.6028/NIST.FIPS.202>, 8 2015. [Online; cit. 18.05.2016].
- [9] NIST. NIST Kicks Off Effort to Defend Encrypted Data from Quantum Computer Threat. <http://www.nist.gov/itl/csd/nist-kicks-off-effort-to-defend-encrypted-data-from-quantum-computer-threat.cfm>, 2016 April. [Online; cit. 18.05.2016].
- [10] OUZAN, S., AND BE'ERY, Y. Moderate-density parity-check codes. *arXiv preprint*, 0911.3262 (2009).
- [11] SHOR, P. W. Algorithms for quantum computation: discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science Proceedings* (1994), IEEE.

- [12] SHOUP, V. NTL: A Library for doing Number Theory. <http://www.shoup.net/ntl/>. [Online; cit. 18.05.2016].
- [13] ZAJAC, P. A note on CCA2-protected McEliece cryptosystem with a systematic public key. *IACR Cryptology ePrint Archive* (2014).

Prílohy

A	Manuál k priloženým programom	II
B	Štruktúra elektronického nosiča	V

A Manuál k priloženým programom

A.1 knižnica uEliece (Programs/*/uEliece/)

Knižnicu je možné skompilovať spustením príkazu **make** v zložke so zdrojovými súbormi. Pri implementáciách pre AVR je potrebné mať nainštalovaný kompilátor **avr-gcc**. Tiež je potrebné v súbore Makefile nastaviť hodnotu premennej **DEVICE** na označenie cieľového mikropočítača.

Do programu, v ktorom plánujeme knižnicu využívať, je potrebné vložiť hlavičkový súbor **uEliece.h** - `#include "uEliece.h"`.

Potom môžeme používať funkcie, deklarované a popísané v **uEliece.h**:

- `uint8_t uEliece_decrypt(...);`
- `uint8_t uEliece_encrypt(...);`

Spôsob použitia funkcií a požadované argumenty sú presne popísané v komentároch pri deklaráciách funkcií. Objekty, generované pri kompilácii knižnice, je potrebné linkovať pri kompilácii výsledného programu.

A.2 x86/uEl-file

Program môžeme skompilovať spustením príkazu **make** v zložke so zdrojovými súbormi. Pre automatické vygenerovanie nového kľúčového páru a jeho použitie môžeme použiť príkaz **make key**. Potom je potrebné nanovo skompilovať program.

Výsledný spustiteľný súbor je konzolová aplikácia, ktorá prijíma 3 argumenty:

1. "-e" pre šifrovanie alebo "-d" pre dešifrovanie.
2. Názov vstupného súboru.
3. Názov výstupného súboru.

Po spustení program načíta vstupný súbor, zašifruje alebo dešifruje ho pomocou kľúčov, ktoré doň boli vložené pri kompilácii a výsledok zapíše do výstupného súboru.

A.3 x86/uEl-keygen

Na kompiláciu generátora kľúčových párov je potrebné mať nainštalovanú knižnicu NTL. Program je vtedy možné skompilovať spustením **make** v zložke so zdrojovými súbormi.

Program je konzolová aplikácia, prijímajúca jeden parameter. Ním sa nastavuje formát výstupných súborov alebo zobrazuje nápoveda:

- `-?` : zobrazenie nápovedy
- `-H` : výstup do textového súboru, kľúč je formátovaný ako hexadecimálne číslo
- `-K` : výstup do textového súboru, kľúč je formátovaný ako vektor hexadecimálnych čísel, oddelených :
- `-C` : výstup do textového súboru, kľúč je formátovaný ako inicializátor poľa v jazyku C. Typ poľa korešponduje s dátovými typmi pre uloženie kľúčov, definovanými v knižnici `uEliece`
- `-B` : výstup do binárneho súboru

Program po spustení náhodne vygeneruje súkromný kľúč, vypočíta k nemu verejný kľúč a kľúče uloží do súborov `uEl_priv.key` a `uEl_pub.key` v adresári, z ktorého bol spustený.

A.4 AVR/Atmega1284P/EncDec

Na kompiláciu testovacieho programu pre AVR a jeho nahranie na mikropočítač je potrebné mať nainštalovaný kompilátor `avr-gcc` a nástroj `avr-objcopy` a `avrdude`. V súbore `Makefile` je potrebné nastaviť premenné `DEVICE` a `AVRDUDE-DEVICE` na označenie cieľového mikropočítača, používané kompilátorom a `avrdude`. Na nahranie programu na mikropočítač je tiež potrebné nastaviť premennú `PROGRAMMER` na označenie použitého programátora.

Potom môžeme program skompilovať volaním príkazu `make` a nahráť ho na mikropočítač volaním `make flash`. Program na šifrovanie a dešifrovanie využíva kľúčový pár vložený do zdrojového kódu prostredníctvom súboru `uEl_keys.h`. Tento súbor je rovnaký, ako používa program `x86/uEl-file`. Pri zmene kľúčového páru je potrebné program nanovo skompilovať a nahráť na mikropočítač.

Program beží na mikropočítači v nekonečnom cykle. Komunikuje prostredníctvom sériového rozhrania mikropočítača `UART0`. Po spustení najprv nastaví komunikačné rozhranie, odošle správu `"Device initialized."`, potom vstúpi do nekonečného cyklu. V každom cykle najprv odošle správu `"Waiting for request..."`. Potom čaká, kým prijme požiadavku vo formáte:

- 1 byte s hodnotou 'E', 'D' alebo 'T' ako označenie módu. E - šifrovanie, D - dešifrovanie, T - testovanie oboma smermi.
- 2 byte reprezentujúce `uint16_t`, vyjadrujúci veľkosť správy na šifrovanie/dešifrovanie
- správa

Po prijatí celej požiadavky program odošle naspäť detaily prijatej požiadavky, vykoná požadovanú akciu, odošle výslednú správu a cyklus spustí od začiatku.

B Štruktúra elektronického nosiča

Pri kompilácii jednotlivých programov je potrebné zachovať štruktúru adresárov.

/ uEliece.pdf	: Elektronická verzia bakalárskej práce
/ Programs /	: Vytvorené programy
AVR /	: Implementácie pre AVR
Atmega1284P /	: Implementácie pre ATmega1284P
EncDec /	: Zdrojový kód testovacieho programu
uEliece /	: Zdrojový kód knižnice μ Eliece
/Keccak /	: Zdrojový kód hash funkcie
x86 /	: Implementácie pre x86
uEl-file /	: Zdrojový kód testovacieho šifrátoru súborov
uEl-keygen /	: Zdrojový kód generátora kľúčového páru
uEliece /	: Zdrojový kód knižnice μ Eliece
Keccak /	: Zdrojový kód hash funkcie