МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Пояснювальна записка до дипломної роботи

магістра
(освітній ступінь)

на тему «Експериментальне дослідження постквантумних методів шифрування та криптографії»

ХАІ.603.6-95ПЗ1.121.168928.20О

Виконав: студент 6 курсу групи № 6-95пз1
Спеціальність 121 – Інженерія програмного забезпечення
(код та найменування)
Освітня програма Хмарні обчислення та Інтернет речей
(найменування)
Хірний Д.О.
(прізвище й ініціали студента)
Керівник Шостак І.В.
(прізвище та ініціали)
Рецензент Мартовицький В.О.
(прізвище та ініціали)

Харків – 2020

Міністерство світи і науки України
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»

Факультет програмної інженерії та бізнесу
(повне найменування)
Кафедра інженерії програмного забезпечення
(повне найменування)
Рівень вищої освіти другий (магістерський)

Спеціальність 121 – інженерія програмного забезпечення
(код та найменування)
Освітня програма хмарні обчислення та Інтернет речей
(найменування)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ _____І. Б. Туркін_____
(підпис) (ініціали та прізвище)
“_____” _____ 2020 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТУ

Хірному Дмитру Олександровичу
(прізвище, ім’я, по батькові)

1. Тема дипломного проекту *Експериментальне дослідження постквантумних методів шифрування та криптографії*
керівник дипломного проекту *Шостак Ігор Володимирович, д.т.н., професор*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “__” ___2020 року № __
2. Термін подання студентом роботи _____
3. Вихідні дані до роботи: *постквантумні методи шифрування та криптографії*
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
огляд та аналіз особливості квантових алгоритмів шифрування, та проблеми квантових обчислень на яких базується постквантовий захист; експериментальне дослідження постквантових алгоритмів шифрування; розробка програмної моделі постквантового алгоритму шифрування та провести її тестування

5. Перелік графічного матеріалу
РПЗ – стор. 71, таблиць – 2 шт., презентація – 19 слайдів

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Шостак І.В., проф. каф. 603		
2	Шостак І.В., проф. каф. 603		
3	Шостак І.В., проф. каф. 603		

8. Нормоконтроль _____ __В.А. Постернакова__ «____» _____2020 р.
(підпис) (ініціали та прізвище)

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту	Строк виконання етапів проекту	Примітка
1	Отримання і затвердження теми диплому	03.09.2019	
2	Аналіз предметної області	04.09.2019	
3	Постановка задачі	20.11.2019	
4	Проведення теоретичних досліджень	22.11.2019	
5	Розробка прототипу ПЗ	02.09.2020	
6	Підготовка пояснювальної записки	22.10.2020	
7	Оформлення пояснювальної записки до дипломного проекту	10.11.2020	
8	Передзахист дипломного проекту	27.11.2020	
9	Захист дипломного проекту	24.12.2020	

Студент _____ Хірний Д.О.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Шостак І.В.
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту містить 71 стор., 1 додаток, 18 джерел.

Об'єкт дослідження – алгоритми захисту від квантових атак.

Предмет дослідження – постквантові алгоритми шифрування.

Метою роботи є дослідження захищеності постквантових алгоритмів шифрування з урахуванням квантових обчислень.

Для досягнення поставленої мети необхідно вирішити ряд завдань: огляд та аналіз особливості квантових алгоритмів шифрування, та проблеми квантових обчислень на яких базується постквантовий захист; експериментальне дослідження постквантових алгоритмів шифрування; розробка програмної моделі постквантового алгоритму шифрування та провести її тестування.

Наукова новизна. Удосконалено постквантовий алгоритм шифрування, який на відміну від існуючих оптимізує використання внутрішньої пам’яті через зменшення розміру початкового файлу програми і розміру секретного ключа, що дає графічну стійкість і тяжкість застосування квантових алгоритмів.

Практична значимість отриманих результатів. В результаті проведених досліджень розроблено програмну модель постквантового алгоритму шифрування та експериментально доведено, що порівняно з існуючими створена програма швидше виконує загальну роботу по створенню ключів, шифруванню та розшифруванню.

ЦИФРОВЕ ШИФРУВАННЯ, АЛГОРИТМ ШОРА, КВАНТОВИЙ КОМП’ЮТЕР, ПОСТКВАНТОВА КРИПТОГРАФІЯ, NTRUENCRYPT

ABSTRACT

Explanatory note to the master's thesis 71 pp., 1 app., 18 sources.

The object of study - algorithms for protection against quantum attacks.

The subject of research - post-quantum encryption algorithms.

The aim of the work is to study the security of post-quantum encryption algorithms taking into account quantum calculations.

To achieve this goal it is necessary to solve a number of tasks: review and analysis of the features of quantum encryption algorithms, and the problems of quantum computing on which post-quantum protection is based; experimental study of post-quantum encryption algorithms; development of a software model of the post-quantum encryption algorithm and its testing.

Scientific novelty. The post-quantum encryption algorithm has been improved, which, unlike the existing ones, optimizes the use of internal memory by reducing the size of the initial program file and the size of the secret key, which gives graphical stability and difficulty of using quantum algorithms.

The practical significance of the results obtained. As a result of the research, a software model of the post-quantum encryption algorithm was developed and it was experimentally proved that in comparison with the existing ones, the created program performs the general work on key generation, encryption and decryption faster.

DIGITAL ENCRYPTION, SHORE ALGORITHM, QUANTUM COMPUTER, POSTQUANTIC CRYPTOGRAPHY, NTRUENCRYPT

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ECC - криптосистеми на еліптичних кривих

NTRUEncrypt - Nth-degree TRUncated polynomial ring.

ВСТУП

У найближчому майбутньому може статися нова промислова революція, так як, все зараз ми чуємо усі її легкі відгуки. Тим самим ми починаємо задумуватися про захист нашої електронної інформації на порозі цього. Із кожним роком продуктивність та потужність комп’ютерів росте усе більше, і питання напрошується саме по собі, а як далеко від нас створення надпотужних квантових комп’ютерів які своєю потужністю у обчисленнях зможуть обійти будь-який захист який ми нині вважаємо дуже сильним. Безпека NTRUEncrypt побудована на протидії зламам шифротексту, коли при поміченій атаці зростає графічна стійкість, також застосування відкритого ключа дає свої переваги[1, 2].

Квантові комп'ютери можуть підірвати цей криптографічний захист. Сьогодні ці машини поки недостатньо потужні, але вони швидко еволюціонують. Можливо, що не пізніше, ніж через десять років - а можливо, і раніше - ці машини зможуть стати загрозою для широко використовуваних методів криптографії. Саме тому дослідники і компанії, що займаються безпекою, наввипередки розробляють нові підходи до криптографії, які зможуть витримати майбутні квантові атаки хакерів [3,4].

Для деяких класів завдань алгоритми, що виконуються на квантовому комп'ютері, здатні досягти меншої тимчасової складності, чим при виконанні на класичному комп'ютері. Використання квантових алгоритмів істотно впливає на відкриту криптографію.

Головною метою є дослідження криптосистеми NTRUEncypt яка у майбутньому може протидіяти квантовим комп’ютерам і сама еволюціонує з роками. Вона має дуже розвинений захист ключа, може захиститися від атак на ключі, і навіть якщо хакер має готовий шифротекст [5, 6].

Об'єкт дослідження – алгоритми захисту від квантових атак.

Предмет дослідження – постквантові алгоритми шифрування.

Метою роботи є дослідження захищеності постквантових алгоритмів шифрування з урахуванням квантових обчислень.

Для досягнення поставленої мети необхідно вирішити ряд завдань: огляд та аналіз особливості квантових алгоритмів шифрування, та проблеми квантових обчислень на яких базується постквантовий захист; експериментальне дослідження постквантових алгоритмів шифрування; розробка програмної моделі постквантового алгоритму шифрування та провести її тестування.

Наукова новизна. Удосконалено постквантовий алгоритм шифрування, який на відміну від існуючих оптимізує використання внутрішньої пам’яті через зменшення розміру початкового файлу програми і розміру секретного ключа, що дає графічну стійкість і тяжкість застосування квантових алгоритмів.

Практична значимість отриманих результатів. В результаті проведених досліджень розроблено програмну модель постквантового алгоритму шифрування та експериментально доведено, що порівняно з існуючими створена програма швидше виконує загальну роботу по створенню ключів, шифруванню та розшифруванню.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Пошта будь-якого власника ПК може бути перехоплена, а колегам нічого не завадить ознайомитися з вашими документами. Шифрування – кодування інформації, після якого її неможливо буде прочитати без спец ключа, – зможе захистити ваші данні від атак. Зараз шифрування стає розумною мірою захисту для усіх тих, хто вдома або на роботі використовує ПК: найкращий спосіб зберегти службову інформацію [7].

Незалежно від того, чи використовується автономна утиліта чи вбудована функція поштової програми, процес шифрування відбувається однаково: данні опрацьовуються по спеціальному алгоритму, у результаті чого з’являється зашифрований текст. Алгоритму для роботи треба отримати від вас одну змінну – ключ, і через це сторонній не зможе розкрити шифр.

1.1 Цифрове шифрування

Існують два основних типи шифрування. Симетричне та асиметричне, інакше назване шифруванням з відкритим ключом. При симетричному шифруванні ви створюєте ключ, пропускаєте із ним файл через програму та робите пересилку результату адресату, а ключ висилаєте роздільно. Запустив ту ж саму шифрувальну програму із отриманим ключем, адресат зможе прочитати повідомлення. Симетричне шифрування не так надійно, як асиметричне, оскільки ключ може бути перехоплено, але через високу швидкість воно використовується у операціях електронної торгівлі .

Іноді два цих підходу використовуються разом. У випадку з HTTPS, наприклад, веб-браузери використовують публічні ключі для перевірки достовірності сайтів і отримання ключа для симетричного шифрування комунікацій.

Асиметричне шифрування складніше, але більш надійне. Для нього потрібні два взаємопов’язаних ключа – відкритий та закритий. Свій відкритий ключ ви повідомляєте усім бажаючим. Він позволяє кодувати данні, але не розкодувати їх. Закритий ключ є тільки у вас. Коли комусь потрібно відіслати вам зашифроване повідомлення, він виконує шифрування, використовуючи ваш відкритий ключ. Отримав повідомлення, ви розшифровуєте його з допомогою свого закритого ключа. За вищу надійність асиметричного шифрування потрібно платити: оскільки обчислення у даному випадку складніші, процедура віднімає більше часу.

Алгоритми, застосовувані для симетричного та асиметричного шифрування, основані на різних принципах. При симетричному шифруванні алгоритм розділяє данні на невеличкі блоки, презентує кожен із них деяким числом, перетворює ці числа по складній математичній формулі, в яку входить ключ, а потім повторює перетворення. В деяких випадках воно виконується декілька десятків разів.

Алгоритм же асиметричного шифрування розглядає текст як одне дуже велике число. Він зводить це число в ступінь, котра також є дуже великим числом, ділить результат на ще одне дуже велике число та вичислює залишок, після чого перетворює цей залишок назад у текст. Шифрувальні програми можуть по-різному використовувати один той самий алгоритм, тому щоб отримувач мав змогу прочитати повідомлення, у нього має бути та ж сама програма, як і у відправника.

І на завершення, останній фрагмент головоломки – це ключі. Вони відрізняються по довжині, і отже, по силі, тому що чим довший ключ, тим більше число можливих комбінацій. Використовуючи 128-бітові ключі, ваш конкретний ключ буде із трильйонів, або 2128 можливими комбінаціями нулів та одиниць [8].

Мета – не дати хакерам за допомогою значних обчислювальних потужностей спробувати вгадати використовувані ключі. Для цього використовуються такі популярні криптографічні методи, як RSA і шифрування за допомогою еліптичних кривих, зазвичай використовують так званні односторонні функції з потайним входом – математичні конструкції, які відносно легко обчислювати в одну сторону для отримання ключів, але дуже складно для зловмисника підробити зворотному інженеру.

Хакери можуть спробувати зламати код, підбираючи всі можливі варіанти ключа. Але зі сторони захисту дуже ускладнюють їм це завдання, використовуючи пари дуже довгих ключів – як 2048-бітної RSA, що використовує ключі довжиною в 617 десяткових чисел. Перебір всіх можливих варіантів приватних ключів займе тисячі, якщо не мільйони років у звичайних комп'ютерів.

1.2 Електронний цифровий підпис

Не можна також не згадати і про цифровий підпис у пост-квантумній криптографії, так як він посягає дуже важливе місце.

Електронний цифровий підпис (ЕЦП) – це блок інформації, який додається до файлу даних автором (підписувачем) та захищає файл від несанкціонованої модифікації і вказує на підписувача (власника підпису) [9].

Для функціонування ЕЦП використовуються 2 ключі захисту (які зберігаються у різних файлах): таємний ключ, який зберігається у підписувача (наприклад, на дискеті, пристрої Touch Memory, Smart-карті), та відкритий ключ, який, як правило, публікується у загальнодоступному або спеціалізованому довіднику.

Для накладання ЕЦП використовується таємний (особистий) ключ, а для його перевірки – відкритий (загальновідомий) ключ.

Алгоритм роботи системи побудовано таким чином, що коли користувач має доступ до відкритого ключа, то він не може відтворити таємний ключ або поставити цифровий підпис – його можна тільки перевірити.

Доцільно зазначити, що таємний (особистий) ключ підписувача є повною особистою власністю підписувача і не надається будь-яким іншім особам (навіть центру сертифікації ключів). Будь-хто може перевірити цифровий підпис, використовуючи тільки відкритий ключ.

1.3 Квантова загроза

Останнім часом широку популярність отримують квантові обчислення. Якщо в класичному комп'ютері найменша одиниця інформації представляється бітом, який може набувати значення або 0, або 1 в одно час, то в квантовому цю роль виконують кубіти. Їх особливість полягає в тому, що кубіт може знаходитися і в змозі 0, і в змозі 1 одночасно. Це і дає квантовим комп'ютерам їх неймовірну обчислювальну потужність. Наприклад, якщо ми розглядаємо чотири біти інформації, то зі всіляких 16 станів ми можемо вибрати лише одно водночас часу. 4 кубіта ж можуть знаходитися в 16 станах одночасно, тобто в суперпозиції, і ця залежність росте експоненційно з кожним новим кубітом.

Якщо в класичному комп'ютері логічні елементи отримують на вхід біти інформації, а на виході видають однозначно певний результат, то в квантовому комп'ютері в якості логічного елементу береться так званий квантовий гейт (quantim gate), який маніпулює значенням цілої суперпозиції. Важливе явище, властиве кубітам, – це заплутаність. Наприклад, маємо два заплутаних кубіта. Вимір стану одного з них допоможе дізнатися інформацію про стан його пари без необхідності якої-небудь перевірки [10].

Слід зазначити, що квантовий комп'ютер – це не заміна звичним нам класичним, оскільки вони швидше лише у виконанні обчислювальних операцій, де необхідно використати всілякі суперпозиції. Так що для перегляду відео на інтернет-сервісі Ютуб використати таку машину зовсім недоцільно. З одного боку, поява квантового комп'ютера – це дуже великий крок для науки і для світу у обчисленнях.

У багатьох сферах науки така машина принесе немало користі (наприклад, при моделюванні), проте для криптографії такий значимий прорив буде критичний. А все тому, що в 1994 році Пітер Шор запропонував квантовий алгоритм, який

дозволяє розкласти число не сто мільйонів років, а за цілком доступний для огляду час.

1.4 Алгоритм Шора

Модифікація цього алгоритму дозволяє вирішити і завдання дискретного логарифмування. Узагальнено метод Шора полягає в зведенні дуже складно вичислюють на класичному комп'ютері завдання до обчислення порядку деякої функції. Якщо розглядати розкладання числа на множники, або завдання факторизації, то в якості тієї самої функції береться $f(x) = a^x (mod\ N)$, де N - число, яке розкладається, а α спеціально підібране значення, взаємно просте з N .Далі по руху алгоритму знаходиться період функції ω, який задовольняє співвідношенню: $f(x+\omega) = f(x)$ і, як наслідок, виконується $a\omega = 1(modN)$. Після знайденого періоду вичислити власного дільника числа N можна за допомогою алгоритму Евкліда: $gcd(a\omega 2, N)$ [11].

Для того, щоб вирішити завдання дискретного логарифмування, тобто, знайти таке k за даними $y = gk$,, необхідно вичислити порядок іншої функції, а саме: $f(x1, x2) = g^{x1} y^{x2}$, де g - твірна групи с числом елементів, рівним q. Період функції представляється парою чисел $(\omega 1, \omega 2) : f(\omega 1 + x1, \omega 2 + x2) = f(x1, x2)$. Тоді рішення задачі дискретного логарифмування матиме вигляд: $k = -\frac{\omega_1}{\omega_2} (mod\ q)$.

Таким чином, в методі Шора можна виділити квантову і класичну частину, при чому завдання квантової частини алгоритму полягає у відшукуванні періоду функції з використанням методу суперпозиції. Недивно, що існування подібних алгоритмів і тенденція до розробки квантових комп'ютерів підштовхнули спеціальні організації до роздумів. Агентством національної безпеки США, наприклад, ще в 2015 році був анонсований план переходу до алгоритмів, стійких

до атаки на квантовому комп'ютері. А в 2016 році NIST США офіційно оголосив про про запуск конкурсу заявок на розробку алгоритмів постквантової криптографії.

Постквантова криптографія не обмежується одним примітивом, насправді, на даний момент розглядаються декілька кандидатів. У їх число можуть, наприклад, входити швидкі криптографічні протоколи на ґратах, схеми на хэш-функциях і криптосистеми, ґрунтовані на некомутативній алгебрі.

1.5 Постквантова криптографія

Це розробка нових видів криптографічних методів, які можна застосовувати з використанням сьогоднішніх класичних комп'ютерів, але які при цьому будуть невразливі перед завтрашніми квантовими.

Одна з ліній оборони – збільшення розмірів цифрових ключів для значного збільшення кількості варіантів, в яких потрібно буде вести пошук перебором. Наприклад, просте подвоєння розміру ключа з 128 до 256 біт збільшить кількість можливих варіантів в чотири рази, які доведеться перебрати квантовій машині, що використовує алгоритм Гровера.

Ще один підхід включає використання більш складних функцій з потайним входом, таких, щоб з ними важко було впоратися навіть потужному квантовому комп'ютера, виконуючому алгоритм Шора. Дослідники працюють над широким спектром підходів, включаючи такі екзотичні, як криптографія на решітках і протокол обміну ключами з використанням суперсингулярного ізогена [9].

Мета досліджень – вибрати один або кілька методів, які потім можна буде широко застосовувати. Національний інститут стандартів і технологій США запустив в 2016 році розробку стандартів постквантового шифрування для урядового використання. Він уже звузив початковий набір заявок з 69 до 26, але каже, що перші чернетки стандартів, швидше за все, з'являться не раніше 2022 року.

Критична важливість цього завдання пов'язана з тим, що технології шифрування глибоко впроваджені в безліч різних систем, тому на їх переробку і впровадження нових алгоритмів потрібно дуже багато часу. У дослідженні національних академій від минулого року відзначено, що на повне позбавлення від одного широко розповсюдженого криптографічного алгоритму, який опинився вразливим, пішло більше 10 років. З огляду на швидкість розвитку квантових комп'ютерів, можливо, у світу залишилося не так вже й багато часу, щоб розібратися з цією новою проблемою безпеки.

Ось чому дослідження пост квантових алгоритмів шифрування дуже важливо для нас зараз і у майбутньому, у роботі буде досліджено пост-квантумну криптосистему NTRUEncrypt, яка має значні переваги перед багатьма нині існуючими системами та може захистити у майбутньому нашу інформацію від різних видів атак [12, 13]. У системі реалізовано алгоритм NTRU-CCA для шифрування та розшифрування ключа [14, 15]

Експериментально доказана перевага використання алгоритму NTRUEncrypt. Нині існуюча програма на 25% швидше виконую загальну роботу по створенню ключів, шифруванню та розшифруванню. Окрім цього, оптимізується використання внутрішньої пам’яті через зменшення розміру початкового файлу програми і розміру секретного ключа. При спробі зламати шифротекст з’являється графічна стійкість і тяжкість застосування квантових алгоритмів.

2 ПЛАНУВАННЯ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ МЕТОДІВ ШИФРУВАННЯ ТА КРИПТОГРАФІЇ NTRU

2.1 Опис криптосистеми

NTRUEncrypt(Nth-degree TRUncated polynomial ring) – це криптографічна система із відкритим ключем, яка раніше мала назву NTRU.

Криптосистема NTRUEncrypt, заснована на ґратчастій криптосистемі, створена у якості альтернативи RSA й криптосистемам на еліптичних кривих (ECC). Стійкість алгоритму забезпечена складністю пошуку найкоротшого вектору ґратки, яка більш стійка до атак, здійснюваних на квантових комп'ютерах. На відміну від своїх конкурентів RSA, ECC, Elgamal, алгоритм використовує операції над кільцем $Z|X|/(X^N - 1)$ усічених многочленів ступені, не перевершує $N - 1$ у формулі 2.1

$$a(X) = a = a_0 + a_1X + a_2X^2 + \cdots + a_{N-2}X^{N-2} + a_{N-1}X^{N-1} \quad (2.1)$$

такий многочлен можливо представляти вектором у формулі 2.2

$$\vec{a}(X) = \vec{a} = \sum_{i=0}^{N-1} a_iX^i = [a_0, a_1, a_2, \dots, a_{N-2}, a_{N-1}] \quad (2.2)$$

як і будь-який молодий алгоритм, NTRUEncrypt ще не досконало був вивчений, хоча й був офіційно затверджений для використання у сфері фінансів.

Додавання елементів у такій групі відбувається при множенні елемент x^n приводиться к 1, x^{n+1} і так далі. При множенні двох елементів кільця $a(x) \times b(x)$ виходить елемент $c(x) = c_0 + c_1x + \cdots + c_{n-1}x^{n-1}$, у якому коефіцієнт c_k обчислюється за формулою 2.3

$$c_k = a_0b_k + a_1b_{k-1} + \cdots + a_kb_0 + a_{k+1}b_{N-1} + a_{k+2}b_{N-2} + \cdots + a_{N-1}b_{k+1} \quad (2.3)$$

Таке кільце отримало назву кільця усічених многочленів. Позначимо його для зручності R. В криптосистемі NTRU усі коефіцієнти многочленів приводяться по модулю цілих чисел p і q. Наприклад, многочлен $13 + 12x + 14x^2 + 7x^3 \; mod \; 3 = 1 - x^2 + x^3$

Зараз перейдемо до написання безпосередньо самого алгоритму. NTRU використовую три постійних параметри: N, p, q. Числом N характеризується розмір вибраних у якості ключів многочленів. Числа p і q не обов'язково мають бути простими, але НОД(p, q) має дорівнювати одиниці Слід відмітити, що параметр p потрібен для визначення інтервалу, якому мають належати усі коефіцієнти многочленів використаних у криптосистемі. А якщо точніше, простів повідомлень L криптосистеми NTRU визначається як $L_m = \left\{ M(x) \in R \middle| \text{ для усіх коефіцієнтів } M \text{ лежачих в } \left[-\frac{p-1}{2}; \frac{p-1}{2} \right] \right\}$.

Таким чином, якщо $N = 11$, а $p = 3$ то ми зможем використовувати у нашій криптосистемі тільки многочлени із коефіцієнтами $\{-1,0,1\}$. Наприклад: $-1 + x + x^3 - x^4 - x^5 + x^{10}$

Після вибору цих трьох основних параметрів треба буде обрати ще три додаткових, котрі прийнято позначати d_f, d_g, d. Ці три параметри служать для визначення набору многочленів $L_f = L(d_f, d_f - 1)$, $L_g = L(d_g, d_g)$, $L_r = L(d, d)$.

Невеличке пояснення, запис виду $L_f = L(d_f, d_f - 1)$ означає, що L_f є набором усіх можливих многочленів кільця R, котрі маємо у якості коефіцієнтів дорівнюють d_f одиниць(1) і $d_f - 1$ одиниць(-1) усі інші коефіцієнти дорівнюють нулю.

Наприклад, многочлен $-1 + x + x^3 - x^4 + x^8$ належить набору $L(3,2)$ тому у нього 3 1кі й 2 -1кі.

Тепер можемо спробувати згенерувати пару – секретний та відкритий ключ. Робиться це таким чином:

- Із набору L_f вибираємо довільний многочлен $f(x)$;
- Із набору L_g вибираємо многочлен $g(x)$;

- Вичислюємо многочлени $f_q(x)$ і $f_p(x)$ такі що $f_p(x) \times f(x) = 1 \; mod \; p$ і $f_q(x) \times f(x) = 1 \; mod \; q$;
- Відкритий ключ визначається як $h(x) = f_q(x) \times g(x) \; mod \; q$;
- Секретний ключ це пара $\Big(f(x), f_p(x)\Big)$.

Описуємо процедуру шифрування у криптосистемі NTRU. Людина1 обирає повідомлення m і перетворює його в многочлен $M(x) \in L_m$, нагадаємо, що коефіцієнти многочленів, що належать L_m лежать у інтервалі $\left[-\frac{p-1}{2}; \frac{p-1}{2}\right]$

Людина1 обирає «осліплюючий» многочлен $r(x) \in L_r$, і використовуючи відкритий ключ Людини2 вичислює $C(x) = p \times r(x) \times h(x) \times M(x) \; mod \; q$. Многочлен $C(x)$ і буде тим самим шифротекстом.

Тепер маємо змогу описати процедуру розшифровки. Отримавши від Людини1 $C(x)$, Людина2 використовує секретний ключ і вичислює $a(x) = f(x) \times C(x) \; mod \; q = f(x) \times p \times r(x) \times h(x) + f(x) \times M(x) \; mod \; q = f(x) \times r(x) \times p \times f_q q(x) \times g(x) + f(x) \times M(x) \; mod \; q = r(x) \times p \times g(x) + f(x) \times M(x) \; mod \; q$.
При цьому Людина2 дуже ретельно слідкує, щоб коефіцієнти отриманого многочлена $a(x)$ лежали у інтервалі $(-\frac{q}{2}, \frac{q}{2}]$

Людина2 обчислює $b(x) = a(x) \; mod \; p = f(x) \times C(x) \; mod \; p = r(x) \times p \times g(x) + f(x) \times M(x) \; mod \; p$ перший доданок виразу $b(x)$ має множник p і отже $b(x) = f(x) \times M(x) \; mod \; p$. Однак це усе стається тільки у тому випадку, якщо при обчисленні $a(x)$ його коефіцієнти були не більше q, саме тому на першому етапі розшифрування і відбувається перевірка того, що усі коефіцієнти лежать у указаному інтервалі.

Вичислив $b(x) \times f_p(x) \; mod \; p = f(x) \times b(x) \times f_p(x) = M$ Людина2 відновлює повідомлення M

Найбільш цікавий момент у цій схемі шифрування-розшифрування я вважаю момент перевірки належності до коефіцієнтів отриманого многочлену $a(x)$ інтервалу $(-\frac{q}{2}, \frac{q}{2}]$. Усі коефіцієнти, як ми бачимо, мають бути не більше q щоб не порушити подільність на p лівої частини суми. Однак чому ми тоді робимо

перевірку інтервалу $(-\frac{q}{2}, \frac{q}{2}]$ а не $(-q, q]$? Зробимо припущення, що $q = 32$. $P = 3$. У результаті приведення по модулю 32 отримали число 18. По модулю 3 це дорівнює нулю. Адже $18 = -14\ mod\ 32$. І саме тому ми вирахували $-14\ mod\ 3$, то отримаємо невірний результат. Відповідно потрібно завжди заздалегідь знати у я кому інтервалі будуть отриманні коефіцієнти. Розробники NTRU стверджують, що для рекомендованих параметрів із вірогідністю рівній майже одиниці коефіцієнти завжди будуть розташовуватися у інтервалі $(-\frac{q}{2}, \frac{q}{2}]$, тому при розшифровці Людина2 призводить умовно отримане число 18 к -14

2.2 Плюси і мінуси NTRU

Отже, такі плюси та недоліки ми можемо побачити у NTRU все зараз. Почнемо ми з плюсів:

- По-перше, це велика швидкість роботи. Виконання операцій шифрування та розшифрування потребує $O(n^2)$ операцій, на відміну від $O(n^3)$ у того ж RSA;
- По-друге, невелике, але збільшення стійкості при фактично тій же довжині ключа, у таблиці (2.1) це зображено.

Таблиця 2.1 – Приблизна оцінка часу злому криптосистем RSA і NTRU

RSA-1024	1012 MIPS-years
NTRUEncrypt N = 163	1014 MIPS-years
RSA 2048	1021 MIPS-years
RSA 4096	1033 MIPS-years
NTRUEncrypt N = 503	1035 MIPS-years

Мінус на даний час ми маємо лише один і дуже незначний. Необхідність використання лише рекомендованих параметрів. Саме ці вимоги визивали суспільні незадоволення під час переходу на еліптичні криві, й сприяло усіляким підозрам про наявність бекдорів. Рекомендовані параметри NTRU зображено у таблиці (2.2).

Таблиця 2.2 – рекомендовані параметри для NTRU

	N	p	q	df	dg	Dr
NTRU167:3	167	3	128	61	20	18
NTRU251:3	251	3	128	50	24	16
NTRU503:3	503	3	256	216	72	55
NTRU167:2	167	2	127	45	35	18
NTRU251:2	251	2	127	35	35	22
NTRU503:2	503	2	253	155	100	65

2.3 Стійкість NTRU

Нехай $\{b_1, b_2, \dots, b_n\}$ – лінійно незалежна система векторів. Ґратами L називають безліч цілочисельних лінійних комбінацій. $L(b_1, \dots, b_2) = \{\sum_{i=1}^{n} x_i b_i : x_1, \dots, x_n \in Z\}$. До прикладу, ґратка породжена парою векторів $b_1 = (2,0)$ і $b_2 = (1,1)$ складається з усіх векторів виду $\begin{pmatrix} 2x + y \\ y \end{pmatrix}, x, y \in Z$ або іншими словами векторів виду $\begin{pmatrix} a \\ b \end{pmatrix}$ таких що $\begin{pmatrix} a \\ b \end{pmatrix} = \begin{pmatrix} 2 & 1 \\ 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix}$

Лобова атака на NTRU ґрунтується на ґратах і пошуку найкоротшого вектору в ґратах. Для розкриття секретного ключа $f(x)$ той, що атакує може побудувати матрицю. Матрицю зображено у формулі 2.4

$$\left(\begin{array}{cccc|cccc} \alpha & 0 & \cdots & 0 & h_0 & h_1 & \cdots & h_{n-1} \\ 0 & \alpha & \cdots & 0 & h_{N-1} & h_0 & \cdots & h_{n-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \alpha & h_1 & h_2 & \cdots & h_0 \\ \hline 0 & 0 & \cdots & 0 & q & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 & 0 & q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 & 0 & 0 & \cdots & q \end{array}\right) \quad (2.4)$$

І згенерувати з рядків цієї матриці ґрати L. Т.к. відкритий ключ Людини2 $h(x) = g(x) \times f^{-1}(x)$ ці ґрати міститиме вектор $t = \big(a \times f(x), g(x)\big)$. Більше того цей вектор являється найкоротшим в ґратах L. Відповідно знаходження такого вектору приведе до знаходження секретного ключа $f(x)$. Завдання пошуку найкоротшого вектору ґрат вважається обчислювально важким завданням. Тимчасову оцінку злому NTRU по методу ґрат можна вичислити по формулі $T = 2^{(0,4N-3,5)}$. Для $N = 251$ це складає приблизно 2^{100}.

Декілька слів про квантові обчислення. Річ у тому, що із появою квантового комп'ютера можна буде реалізувати алгоритм Шора, який дозволяє вирішити задачу факторизації, а також дискретного алгоритму. Звісно, що у світі цього RSA, DSA й інші алгоритми становляться зовсім марні.

Декілька слів про стійкість. Річ у тому, що NTRU як і RSA не гарантують стійкості у випадку доказу нерівності $P \neq NP$, що пояснюється віднесенням задачі до класу NP навіть у тому випадку, якщо маємо хоча б один важко розв'язний варіант або кажучи по гіршому випадку, у той час як інші варіанти можуть мати легке рішення. Відповідно ніхто не зможе дати гарантії, що навіть якщо не буде доведено що $P \neq NP$, хакеру не попадеться більш легкий варіант задачі і злам не буде можливим за поліноміальний час. Деяка задачі засновані на ґратках мають доказаний взаємозв'язок стійкості у середньому й найгіршому випадку. Це дає надію на те, що у майбутньому поліноміальну стійкість при виконання умови $P \neq NP$. Саме ці дві відмінності роблять NTRU настільки несхожим на своїх попередників.

2.4 Відомі уразливості системи

Коротко пробіжимося по уразливості системи й розглянемо найбільш успішні атаки.

Брут форс. При зламі грубою силою, основне завдання суперника підібрати секретний ключ Людини2, тобто многочлен $f(x)$. Противнику відомо, що многочлен $f(x)$ довжиною N має d_f одиничних коефіцієнтів і $(d_f - 1)$ коефіцієнтів -1. Підбір такого многочлену треба перевіряти $\binom{N}{d_f}\binom{N-d_f}{d_f-1}$ варіантів. Для $N = 251$ і $d_f = 50$ цей вираз дорівнює 3×10^{100}. Порівнюючи дану оцінку з оцінкою важкості рішення задачі пошуку найкоротшого шляху вектору ґратки можна зробити висновок про те, що як і у випадку із RSA брут-форс ключа є не самою вдалою атакою проти NTRU.

Атака зустріч посередині. Було запропоновано варіант атаки посередині, яка для успішного злому секретного ключа потребує $\binom{N/2}{d/2}/\sqrt{r}$ часу і рівно стільки ж місця на ЖД, також треба замітити, що r-ціле число не більше N. Атаки такого типу називають зустріч по середині, тому що дозволяють розміняти час потрібний для обчислень на пам'ять необхідну для зберігання тимчасових даних. Атака має вигляд наступним чином. Із визначення відкритого ключа $h = f_q \times g \; mod \; q$ слідує рівність $h = f_q \times g \; mod \; q$. При цьому хакер представляє f як конкатенацію двох многочленів довжиною $\frac{N}{2}$. $f(x) = f_1 \| f_2$, то $g = h \times (f_1 \| f_2) = f_1 \times h + f_2 \times h \; mod \; q$. Ми знаємо, що многочлен g складається із коефіцієнтів $\{1,0,-1\}$ для випадку $p = 3$ або коефіцієнтів $\{1,0\}$ для випадку $p = 2$, тобто іншими словами $f_1 \times h = \{1,0\} - f_2 \times h$.

Виходячи із цього факту хакер може діяти по наступному алгоритму(для випадку $p = 2$):

- Вибирається число k. І записується 2^k корзин, у яких будуть зберігатися відповідні кандидати f_1 і f_2. Збільшення числа k приводить до зменшення часу алгоритму, але к збільшенню необхідної для злому пам'яті.
- Хакер перебирає усі варіанти многочленуf_1 довжина якого дорівнює $\frac{N}{2}$ має $\frac{d}{2}$ 1ки.Такий перебір потребує $\binom{N/2}{d/2}$ ітерацій.
- Кожен отриманий многочлен хакер поміщує у корзину наступним чином: f_1 записується у корзину із номером, який складається із найбільш значущих біт перших k коефіцієнтів $f_1 \times h \; mod \; q$. Наприклад: нехай $N = 4$,$q = 8$. Тоді многочлен із коефіцієнтами $\{7,2,3,5\}$ буде поміщено у корзину із номером $\{1,0,0,1\}$. А многочлен із коефіцієнтами $\{6,4,3,1\}$ у корзину$\{1,1,0,0\}$.
- Після цього хакер починає перебирати варіанти многочлену f_2, що містить $\frac{d}{2}$ 1ки. Такий перебір також займе $\binom{N/2}{d/2}$ ітерацій.
- Під час перебору многочлену f_2 хакер записує отриманий многочлен не в одну корзину, а у декілька по наступному принципу: по-перше він записує f_2 у ту корзину котра відповідає старшим бітам многочлену $-f_2 \times h \; mod \; q$, а по-других додаючи до кожного коефіцієнта із $f_2 \times h$ одиниці і отримуючи новий варіант корзин записує многочлен f_2 і в них. Так наприклад многочлен $-f_2 \times h \; mod \; q = \{6,2,1,5\}$ записується лише у корзину$\{1,0,0,1\}$, а многочлен $-f_2 \times h \; mod \; q = \{7,2,3,5\}$ у корзини $\{1,0,0,1\}$, $\{1,0,1,1\}$, $\{0,0,0,1\}$, $\{0,0,1,1\}$.
- Якщо у корзині у котру хакер записує многочлен f_2 вже зберігається f_1, тобто для цих многочленів виконується умова $f_1 \times h = \{1,0\} - f_2 \times h \; mod \; q$. Саме тому вони рахуються хорошими кандидатами для встановлення f. Хакер вичислює $(f_1 \| f_2) \times h \; mod \; q$ і якщо у результаті отримується многочлен із коефіцієнтами $\{0,1\}$, що відповідає многочлену g, значить секретний ключ f знайдено.

Таким чином, якщо простір ключів криптосистеми NTRU має розмір $\binom{N}{d}$, то пошук ключа по методу зустріч по середині потребу перебирати усього $\binom{N/2}{d/2}$ варіантів.

Так для NTRU із параметрами $N = 251, p = 2, d = 35$ простір має розмір $\approx 2^{140}$, а атака зустріч по середині потребує перебирати $\approx 2^{70}$ варіантів(використовуючи при цьому 2^{70} пам'яті). Тобто для того щоб гарантувати рівень стійкості 2^x необхідно вибирати параметри криптосистеми NTRU із простором ключів 2^{2x}.

А зараз ми розглянемо атаку із підібраним шифротекстом. Даний тип атаки найнебезпечніший із практичної точки зору. Пам'ятаємо, що при розшифровці повідомлення Людина2 вичислює вираз у формулі 2.5.

$$a(x) = f(x) \times c(x)\ mod\ q = p \times r(x) \times g(x) + f(x) \times c(x)\ mod\ q \qquad (2.5)$$

Пам'ятаємо також, що усі коефіцієнти отриманого многочлену лежать у інтервалі $(-\frac{q}{2}, \frac{q}{2}]$. Це означає що $p \times r(x) \times g(x) + f(x) \times c(x)\ mod\ q = p \times r(x) \times g(x) + f(x) \times c(x)$, тобто многочлен до приведення по модулю q відповідає многочлену після проведення по модулю q. Атака із підібраним шифротекстом на NTRU заключено у створення такого многочлену $a(x)$, для котрого $a(x)\ mod\ q \neq a(x)$. Атака ґрунтується на наступному:

- Хакер створює шифротекст $C(x) = y \times h(x) + y$ (де y ціле число, а $h(x)$ відкритий ключ Людини2) і відправляє Людині2.
- При спробі розшифрувати повідомлення Людина2 вичислює $a = f(x) \times C(x)\ mod\ q = y \times f(x) \times h(x) + y \times f(x)\ mod\ q = y \times g(x) + y \times f(x)\ mod\ q$ многочлени $g(x)$ і $f(x)$ мають коефіцієнти $\{-1,0,1\}$, то коефіцієнти многочлена a належать множині $\{0, y, -y, 2y, -2y\}$. Виходить, що якщо хакер вибрав y, таким що $y < \frac{q}{2}$ і $2y > \frac{q}{2}$, то при

зведенні $a(x)$ по модулю q, змінюються лише ті елементи многочлена $a(x)$ коефіцієнти у яких дорівнюють $\pm 2y$.

- Представимо тепер, що і-й коефіцієнт $a_i = 2y$, тоді $a(x)\ mod\ q = y \times g(x) + y \times f(x) - q \times x^i$ і значить остаточне повідомлення після розшифровки набуває вигляд $f_p(x) \times a(x) = y \times f_p(x) \times g(x) + y \times f(x) \times f_p(x) - q \times x^i \times f_p(x)\ mod\ p$. Якщо хакер вибирає y поділеним націло на p, то у результаті виходить многочлен $z(x) = -q \times f_p(x) \times x^i\ mod\ p$.
- Для визначення секретного ключу Людини2, залишилось лише вирахувати $-q \times z^{-1}(x) \times x^i\ mod\ p$.

Застосовуючи дану схему атаки, хакер може відновити секретний ключ із ймовірністю $P = 0{,}13$ або, грубо кажучи, для відновлення секретного ключа хакеру потрібно відправити усього порядком десяти підібраних шифротекстів.

Зараз розглянемо захист від атаки із підібраним шифротекстом. Значить для того, що захистити NTRU від подібного типу атак, нам рекомендовано використовувати систему NTRU із схемою доповнення FROST [11, 12].

Тобто, при шифруванні по методу NTRU-FROST Людина1 як і у звичайній схемі NTRU вичислює многочлен відкритого тексту $m(x)$. Доповнюючи многочлен випадковим набором із k біт R, Людина 1 вичислює $r(x) = H(\ m(x)\|\mathrm{R})$, де $H(x)$ – криптографічно сильна хеш-функція.

Далі для отримання шифротексту як і у звичайній схемі NTRU Людина1 формує многочлен $c(x) = r(x) \times h(x) + m(x)\ mod\ q$. Отримав шифротекст, Людина2 відновлює повідомлення $m(x)$, і вичислює $H(\ m(x)\|\mathrm{R})$. Потім Людина2 вичислює $H(\ m(x)\|\mathrm{R}) \times h(x) + m(x)\ mod\ q$ і порівнює між собою отримане значення $c(x)$. Якщо $H(\ m(x)\|\mathrm{R}) \times h(x) + m(x)\ mod\ q = c(x)$ тоді Людина2 приймає повідомлення, інакше воно відкидає його.

3 ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ МЕТОДІВ ШИФРУВАННЯ ТА КРИПТОГРАФІЇ NTRU

Для реалізації в роботі обрані NTRU алгоритми та схеми, які є фіналістами першого раунду конкурсу NIST, який проводиться для відбору пост квантових алгоритмів [16].

3.1 NTRU-KEM

Алгоритми шифрування тісно пов’язані з алгоритмами інкапсуляції. Алгоритм інкапсуляції – це алгоритм несиметричного шифрування, який зазвичай шифрує постійне повідомлення і результат шифрування потім застосовує в якості ключа для симетричного шифрування.

У наступних пунктах будуть визначатися підтримувані схеми інкапсуляції ключів із відкритим ключем, а також будуть розглянуті тринарні ключі для більш детального поглиблення[17].

Схема NTRU-KEM складається із п’яти операцій. Генерація ключа, перевірка пари ключів, перевірка відкритого ключа, шифрування і дешифрування. Ці операції зазвичай визначаються в цьому реченні як операції, параметризовані компонентами, без прийняття будь-яких конкретних варіантів вибору компонентів. Дійсно у роботі не буде описано усі операції схеми NTRU-KEM, так як вони займуть дуже багато пояснень, тож у роботу буде занесено лише найважливіші і найцікавіші із існуючих операцій схеми.

3.1 1 Генерація ключів

Пара ключів повинна бути сформована із використанням наступного або математично еквівалентного набору кроків. Звертаю увагу, що наведений нижче алгоритм виводить тільки значення f і H. У деяких додатках може бути бажано зберегти ці значення $f - 1$ і g теж. Цей стандарт не визначає формат виведення ключа до тих пір, поки він є однозначним.

Компоненти: параметри N, q, p, dF, D, g.

Вхідні дані: рядок s як джерело випадало.

Вихід: пару ключів, що складається із закритого ключа і відкритого ключа h.

Операція: пара ключів повинна бути розрахована за допомогою наступної або еквівалентної послідовності кроків:

- Встановіть многочлен $F \coloneqq 0$;
- Вмикаємо генератор поліномів фіксованого навантаження;
- Обчислити многочлен $f \coloneqq 1 + p \times F$;
- Обчислити поліном $f - 1$ в кільці. Якщо $f - 1$ не існує, перейдіть до першого кроку
- Викликаємо генератор поліномів фіксованого навантаження із вхідними даними N, D, g, s для отримання a многочлену G;
- Обчислити многочлен $h \coloneqq f - 1 \times g \times p$ в кільці;
- Вихід F, h.

3.1.2 Операція інкапсуляції

Опис операції інкапсуляції. Операція інкапсуляції інкапсулює кільцевий елемент. Він не перевіряє достовірність цього елемента. Виконавець повинен забезпечити достатню ентропію в цьому елементі на рівні протоколу.

Компоненти:

- Довжина буфера кодування, bufferLenBits;
- Повідомлення m, яке являє собою тринарний поліном;
- Відкритий ключ h;
- Рядок s, як джерело впадковості.

Результатом виконання алгоритму є ,зашифрований текст e – , який формується як поліном (кільцевий елемент).

Операція така, що шифротекст e повинен бути розрахований за наступною або еквівалентною послідовності кроків:

- Використовуйте функцію генерації поліномів фіксованим навантаженням з s і параметрами N, dr для отримання r;
- Обчислити $R = r \times h \; mod \; q$;
- Обчисліть шифротекст, як $e = R + M \; mod \; q$;
- Вихід e.

3.1.3 Операція декапсуляції

Компоненти відсутні.

Вхідні дані:

- Шифротекст e, що представляє собою многочлен ступеня $N - 1$;
- Закритий ключ F;
- Відкритий ключ H.

Вихід: елемент із кільця поліномів.

Операція: повідомлення m має бути розраховане за наступною або и теквівалентною послідовністю кроків:

- Обчислити $f = PF + 1$;
- Обчислити $cm = f \times e \, mod \, p$;
- Віхдний СМ, як розшифровка повідомлення М.

3.2 Алгоритми шифрування із відкритим ключем(NTRU-CCA)

Розглянемо деталі реалізації для алгоритму NTRU-CCA.

3.2.1 Алгоритми шифрування

Компоненти:

- Довжина вхідного повідомлення m, lLen в байтах;
- Довжина b, sLen в байтах;
- Функція генерації тринарних поліномів із фіксованоим навантаженням;
- Кількість бітів відкритого ключа до хешу, pkLen;
- Довжина буфера кодування, bufferLenBits.

Вхід:

- Повідомлення m, що представляє собою октетний рядок довжиною l октетів
- Відкритий ключ h

Вихід: зашифрований текст e, який є кільцевим елементом, або « повідомлення занадто довге»

Шифротекст e повинен бути розрахований за наступною або еквівалентною послідовності кроків:

- Якщо $lLen > maxLen$, виведіть «message too long» і зупиніться;
- Сформуйте octet string p0, що складається з 0 байтів, повторених ($maxMsgLenBytes + 1 - lLen$) разів;
- Форма octet string M of length bufferLenBits/8 як ||P0||б|| octL;
- Перетворення M to a bit string Mbin using OS2BSP. Проаналізувати 8lLen polynomial;
- Навпіл sLen з S в якості salt b. розбираємо b як ступінь полінома 8sLen.
- Обчислити $m = Mbin + b * X N - 8sLen - 1$
- Перетворіть відкритий ключ h в бітовий рядок bh за допомогою RE2BSP. Сформуйте бітовий рядок bhtrunc by

Беручи dserші pkLen біти bh. Перетворіть bhTrunc в octet string hTrunc довжиною pkLen/8 за допомогою BS2OSP. Із sData як octet string m

- Використовуйте функцію генерації тринарних поліномів фіксованого навантаження з початковим sData і параметрами N, dr для отримання r;
- Обчислити $R = r \times h \, mod \, q$;
- Використовуйте функцію генерації тринарних поліномів з фіксованим навантаженням з R і обираємо.

Параметри для отримання поліноміальної маски:

- Форма m' шляхом поліноміального складання m і маски mod P;
- Якщо число 1S, або-1s, або 0s в m' менше d m0, відкиньте m' і поверніться до першого кроку;
- Обчислити шифротекст як $e = R + m'MOD \, q$;
- Вихід e.

3.2.2 Алгоритми розшифрування

Компоненти:

- Довжина вхідного шифротексту c, cLen в байтах;
- Довжина salt b, sLen в байтах;
- Функція генерації поліномів з фіксованим навантаженням;
- Кількість бітів відкритого ключа до хешу, pkLen;
- Нижня межа A;
- Мінімальне навантаження d m0;
- Максимальна довжина повідомлення maxMsgLenByte;
- Шифротекст e, що представляє собою многочлен ступеня N-1;
- Закритий ключ F;
- Відкритий ключ h.

Вихід: повідомлення m, яке є octet string, або «Fail».

Операція: повідомлення m має бути розраховане за наступною або еквівалентною послідовністю кроків:

- Розраховуємо:
  - $nLen = ceil\,[\,N/\,8]$ – число octet, необхідних для зберігання N бітів;
  - $bLen = db\,/\,8$ – довжина в октетах випадкових даних;
  - $maxLen = nLen - 1 - lLen - bLen$ – максимальна довжина повідомлення.
- Обчислити $f = PF + 1$;
- Розшифрувати зашифрований текст e за допомогою обраного примітиву дешифрування NTRU з входами e і f, щоб отримати розшифрований поліном cm';
- Якщо число 1s, або -1s, або 0s в ci менше d m0, встановіть значення «fail» рівним 1;
- Обчисліть значення z для $r \times h, cR = e - cm'\ mod\ q$;

- Використовуйте функцію генерації тринарних поліномів з фіксованим навантаженням із початковим значенням cR і вибраними параметрами для отримання поліноміальної маски;
- Форма cM поліноміальне віднімання cm' і маска модулю P;
- Розібрати cM наступним чином:
  - Перші lLen octets являють собою довжину повідомлення. Перетворіть значення, що зберігається в цих октетах, у довжину повідомлення кандидата cl. Якщо $cl > maxMsgLenBytes$, встановіть fail = 1 і встановіть cl = maxL.
  - Наступні октети cl є кандидатом повідомлення cm. Решта октетів повинні бути рівні 0. Якщо це не так, встановіть fail = 1.
  - Останні bLen octet salt cb.
- Перетворіть відкритий ключ h в бітовий рядок bh за допомогою RE2BSP. Сформуйте бітовий рядок bhTrunc by.

Беручи перші pkLen біти bh. Перетворіть bhTrunc в octet string hTrunc довжиною pkLen/8 за допомогою BS2OSP. Із sData як octet string cm.

- Використовуйте функцію генерації тринарних поліномів з фіксованим навантаженням з sData і параметрами N, dr для отримання cr;
- Обчислити $cR' = h \times CR \; mod \; q$;
- Якщо $cR' != cR$, set fail = 1;
- Якщо fail = 1, виведіть «fail». В іншому випадку виведіть cm як розшифроване повідомлення m.

3.2.3 Перевірка ключів

Розглянемо метод перевірки пари ключів. Метод перевірки пари ключів визначає, чи відповідає потенційна пара відкритий/заритий ключ обмеженням для пар ключів, створюваних певним методом генерації ключів.

Наступний метод перевірки ключа відповідає операції генерації ключа описаної в 3.1.1.

Компоненти:

- Параметри N, q, df, D, g.

Вхідні дані:

- Компонент закритого ключа F;
- H – відкритий ключ.

Вихід: «valid» або «invalid».

Операція:

- Перевірте, що h – це многочлени ступеня не більше N-1. Якщо ні, виведіть «invalid» і зупиніться;
- Перевірити, що всі коефіцієнти h лежать в діапазоні $[0, q - 1]$. Якщо будь-які коефіцієнти лежать поза цим діапазоном, виведіть «invalid» і зупиніться;
- private_key_blob перетворити в многочлен F;
- Перевірити, що F тринарний із dF + 1 1 і dF – 1s. якщо немає, то вихід «invalid» і зупиніться;
- Перевірте, що F - многочлен ступеня не більше N-1. Якщо ні, виведіть «invalid» і зупиніться;
- Встановіть $f = 1 + 3\,F\ mod\ q$;
- Встановіть $g = f \times h\ mod\ q$;
- Перевірити, що G тринарний із $(d\ g + 1)\ 1s$ і $d\ g - 1s$. якщо це не так, вихід «invalid» і зупинка;
- Вихід «valid».

Зараз розглянемо повну перевірку відкритого ключа. Повний метод перевірки відкритого ключа визначає, чи задовольняє запропонований відкритий

ключ визначенню відкритого ключа і чи задовольняє він яким-небудь додатковим обмеженням, що накладаються генератором даної пари ключів. Такі методи забезпечують найвищу ступінь впевненості безпеці. Наприклад, для ключів, згенерованих за допомогою операції генерації ключів описаної раніше у роботі, повна перевірка відкритого ключа доведе, що $h = f - 1\ g\ mod\ q$, де $f = 1 + PF$ і F, G мають dF, D, g, 1S відповідно. В даний час не існує відомих методів, що забезпечують повну валідацію відкритого ключа для вище вказаних схемах в цьому стандарті.

Далі часткова валідація і тести правдоподібності, а почнемо з огляду. Метод часткової перевірки відкритого ключа визначає з певним ступенем впевненості, чи відповідає кандидатний відкритий ключ деяким властивостям відкритого ключа. Як і у випадку з повними методами перевірки відкритого ключа, часткові методи перевірки відкритого ключа можуть бути інтерактивними або неінтерактивними. Цей стандарт підтримує тільки неінтерактивні методи.

Неінтерактивні методи для відкритих ключів LBP-PKE, які не вимагають наявності свідка, називаються тестами правдоподібності.

Ця назва відображає той факт, що при дослідженні тільки відкритого ключа тести тільки визначають, чи є відкритий ключ правдоподібним, але не обов'язково є він дійсним. Тести правдоподібності можуть виявити ненавмисні помилки з розумною ймовірністю, хоча і не з упевненістю.

І можу зробити деяку примітку – існують і інші способи виявлення ненавмисних помилок; контрольна сума на ключі може бути використана для виявлення помилок зберігання і передачі, а підпис на сертифікаті, швидше за все, не пройде перевірку, якщо відкритий ключ буде змінений. Перевірки в цьому пункті забезпечують додатковий рівень впевненості

ВИСНОВКИ

У ході виконання дипломної роботи було досліджено проблему захисту від квантових комп’ютерів. Були розглянуті алгоритми захисту такі як: NTRU-KEM, NTRU-CCA. Також були розглянуті особливості квантових алгоритмів, та проблеми квантових обчислень на яких базується постквантовий захист.

Були проведені детальні аналізи криптосистеми - NTRUEncrypt, її переваги та недоліки перед конкурентними алгоритмами, у ході свого дослідження було зрозуміло, що дана криптосистема знаходиться попереду перед іншими системами та алгоритмами шифрування, і є сама по собі достатньо новою і оновлюється із кожним роком.

Реалізовано алгоритм було за допомогою мови C [18], та розглянуто шляхи майбутніх модифікацій самого алгоритму.

Підведемо підсумки. Що до захищеності даної системи: NTRU, пристосована для захисту ключа від майже усіх нині можливих атак. Стійкість алгоритму забезпечена складністю пошуку найкоротшого вектору ґратки, яка більш стійка до атак, здійснюваних на квантових комп’ютерах, була створена для заміни своїх попередників RSA та еліптичних кривих.

Висока швидкість і компактність ключів, оптимізується використання внутрішньої пам’яті через зменшення розміру початкового файлу програми і розміру секретного ключа. При спробі зламати шифротекст з’являється графічна стійкість і тяжкість застосування квантових алгоритмів.

.

ПЕРЕЛІК ПОСИЛАНЬ

1. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 1–23. Springer, Heidelberg (May / Jun 2010).

2. Hoffestin J., Silverman J., Fischlin: Optimisation for NTRU, Public-Key Cryptograhpy and Computional numver theory, DeGruyter (Sept 11-15, 2000)

3. Bruinderink, L.G., Pessl, P.: Differential fault attacks on deterministic lattice signatures. IACR TCHES 2018(3), 21–43 (2018), URL: https://tches. iacr.org/index.php/TCHES/article/view/7267.

4. Poddebniak, D., Somorovsky, J., Schinzel, S., Lochter, M., R¨osler, P.: Attacking deterministic signature schemes using fault attacks. Cryptology ePrint Archive, Report 2017/1014 (2017), URL: http://eprint.iacr.org/ 2017/1014.

5. Blake-Wilson, S., Menezes, A.: Unknown key-share attacks on the stationto-station (STS) protocol. In: Imai, H., Zheng, Y. (eds.) Public Key Cryptography (PKC'99). Lecture Notes in Computer Science, vol. 1560, pp. 154–170. Springer (1999).

6. Menezes, A., Smart, N.P.: Security of signature schemes in a multi-user setting. Des. Codes Cryptogr. 33(3), 261–274 (2004).

7. Jackson, D., Cremers, C., Cohn-Gordon, K., Sasse, R.: Seems legit: Automated analysis of subtle attacks on protocols that use signatures. Cryptology ePrint Archive, Report 2019/779 (to appear at ACM CCS'19) (2019), URL: https://eprint.iacr.org/2019/779.

8. Menezes A. J., van Oorschot P. C., Vanstone S. A.: Handbook of Applied Cryptography. – CRC Press, 1996.

9. Cantero, H., Peter, S., Bushing, Segher: Console hacking 2010 – PS3 epicfail. 27th Chaos Communication Congress (2010), URL: https://www.cs.cmu.edu/~dst/GeoHot/1780_27c3_console_hacking_2010.pdf.

10. Квантовые сети: перспективы и сложности реализации, URL: https://habr.com/ru/company/vasexperts/blog/428740/.

11. Feynman R.: Simulating Physics with Computers International Journal of Theoretical Physics – Springer Science+Business Media, p. 467–488, (1982).

12. Laarhoven T., Mosca M., van de Pol J.: Finding shortest lattice vectors faster using quantum search. Designs, Codes and Cryptography. 1–26 (2014).

13. Poddebniak, D., Somorovsky, J., Schinzel, S., Lochter, M., R¨osler, P.: Attacking deterministic signature schemes using fault attacks. Cryptology ePrint Archive, Report 2017/1014 (2017), URL: http://eprint.iacr.org/ 2017/1014.

14. Bruinderink, L.G., Pessl, P.: Differential fault attacks on deterministic lattice signatures. IACR TCHES 2018(3), 21–43 (2018), URL: https://tches. iacr.org/index.php/TCHES/article/view/7267.

15. Blake-Wilson, S., Menezes, A.: Unknown key-share attacks on the stationto-station (STS) protocol. In: Imai, H., Zheng, Y. (eds.) Public Key Cryptography (PKC'99). Lecture Notes in Computer Science, vol. 1560, pp. 154–170. Springer (1999)

16. Post-Quantum Cryptography. URL: https://csrc.nist.gov/Projects/Post-Quantum-Cryptography/Round-1-Submissions (2018).

17. Gorbenko D., Kachko O.G., Gorbenko Y.I., Stelnik I.V., Kandy S.O., Yesina M.V: Methods for constructing system-wide parameters and keys for NTRU PRIME UKRAINE 5 – 7 stabvility levels. Product form // Radiotekhnika : All-Ukr. Sci. Interdep. Mag. – 2018. - №195. - P.5-16.

18. Shildt H.: C Programming, the Complete Reference, 4th edition. Williams M. (2010).

ДОДАТОК А

Вихідний код програми

```
#include <stdlib.h>
#include <stdint.h>
#include "crypto_stream_salsa20.h"
#include "rng.h"
#include "shred.h"

#define RAND_LEN_BYTES (4096)

static int init = 0;
static unsigned char key[crypto_stream_salsa20_KEYBYTES];
static unsigned char nonce[crypto_stream_salsa20_NONCEBYTES] = {0};
static unsigned char randpool[RAND_LEN_BYTES];
static uint16_t randpos = RAND_LEN_BYTES;


void
fastrandombytes(unsigned char *r, unsigned long long rlen)
{
  unsigned long long n=0;
  uint8_t i;
  if(!init)
  {
    randombytes(key, crypto_stream_salsa20_KEYBYTES);
    init = 1;
  }
  crypto_stream_salsa20(r,rlen,nonce,key);

  // Increase 64-bit counter (nonce)
  for(i=0;i<8;i++)
    n ^= ((unsigned long long)nonce[i]) << 8*i;
  n++;
  for(i=0;i<8;i++)
    nonce[i] = (n >> 8*i) & 0xff;
```

```
}


void rng_cleanup()
{
  if(init)
  {
    init = 0;
    shred(key, crypto_stream_salsa20_KEYBYTES);
    shred(nonce, crypto_stream_salsa20_KEYBYTES);
    shred(randpool, crypto_stream_salsa20_KEYBYTES);
  }
}


void
rng_init()
{
  fastrandombytes(randpool, RAND_LEN_BYTES);
  randpos = 0;
}


void
rng_uint16(uint16_t *r)
{
  if(randpos >= (RAND_LEN_BYTES - sizeof(uint16_t)))
  {
    fastrandombytes(randpool, RAND_LEN_BYTES);
    randpos = 0;
  }
  *r = (uint16_t)(randpool[randpos++] & 0xff) << 8;
  *r |= (uint16_t)(randpool[randpos++] & 0xff);

  return;
}
```

```
void
rng_uint64(uint64_t *r)
{
  if(randpos >= RAND_LEN_BYTES - sizeof(uint64_t))
  {
    fastrandombytes(randpool, RAND_LEN_BYTES);
    randpos = 0;
  }
  *r  = ((uint64_t)(randpool[randpos++] & 0xff)) << 070;
  *r |= ((uint64_t)(randpool[randpos++] & 0xff)) << 060;
  *r |= ((uint64_t)(randpool[randpos++] & 0xff)) << 050;
  *r |= ((uint64_t)(randpool[randpos++] & 0xff)) << 040;
  *r |= ((uint64_t)(randpool[randpos++] & 0xff)) << 030;
  *r |= ((uint64_t)(randpool[randpos++] & 0xff)) << 020;
  *r |= ((uint64_t)(randpool[randpos++] & 0xff)) << 010;
  *r |= ((uint64_t)(randpool[randpos++] & 0xff));

  return;
}
```

Фрагмент коду 1 – Випадкова генерація байтів

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "param.h"
#include "poly.h"
#include "../common/fastrandombytes.h"
#include "../common/crypto_hash_sha512.h"


/*
 * memory requirement: 6 ring elements
 */
void
keygen(
```

```
        uint16_t  *F,     /* output secret key f */
        uint16_t  *g,     /* optional output secret key g */
        uint16_t  *h,     /* output public key h */
        uint16_t  *buf,
    const PARAM_SET *param)
{
    int16_t     i;
    uint16_t    *f;
    uint16_t    *f_inv;
    uint16_t    *localbuf;

    f           = buf;
    f_inv       = f     + param->padN;
    /* three ring elements for karatsuba */
    localbuf    = f_inv + param->padN;

    do{
        /* generate f = pF+1 until f is invertible mod 2*/
        trinary_poly_gen(F, param->N, param->d);
        for (i=0;i<param->N;i++)
            f[i] = param->p*F[i];
        f[0]++;
    }while (ntru_ring_inv(f, param->N, localbuf, f_inv) == -1);


    /* compute f^-1 by lifting f_inv mod 2 to f_inv mod q*/
    ring_lift_inv_pow2(f_inv, f, param, localbuf);

    /* generate g*/
    trinary_poly_gen(g, param->N, param->d);

    for (i=0;i<param->N;i++)
    {
        f[i] = f[i] & 0x7FF;
        g[i] = g[i] & 0x7FF;
    }

    /* compute h = f^-1*g */
```

```
    ntru_ring_mult_coefficients(f_inv, g, param, localbuf, h);

    memset(buf, 0, sizeof(uint16_t)*param->padN*6);
    return;
}

/*
 * memory requirement: 5 ring elements
 * checking if h = g/f
 */
int check_keys(
    const uint16_t  *F,
    const uint16_t  *g,
    const uint16_t  *h,
          uint16_t  *buf,
    const PARAM_SET *param)
{
    int16_t  i;
    uint16_t *f, *grec, *localbuf;

    memset(buf, 0, sizeof(uint16_t)*param->padN*5);
    f           = buf;
    grec        = f     +   param->padN;
    localbuf    = grec  +   param->padN;

    for (i=0;i<param->N;i++)
        f[i] = F[i]*param->p;
    f[0]++;
    ntru_ring_mult_coefficients(f, h, param, localbuf, grec);

    for(i=0;i<param->N;i++)
    {
        if (grec[i]!=g[i])
        {
            printf("checking keys error for %dth coefficients: %d %d\n", i,
grec[i], g[i]);
            return -1;
        }
```

```
    }

    memset(buf, 0, sizeof(uint16_t)*param->padN*5);
    return 0;
}

/* check if message is a valid trinary poly for kem */
int
check_m (
    const uint16_t *m,
    const uint16_t N)
{
    uint16_t i;
    for(i=0;i<N;i++)
    {
        if(m[i]!=1 && m[i]!=65535 && m[i]!=0)
        {
            printf("checking message error for %dth coefficients: %d \n", i,
m[i]);
            return -1;
        }
    }
    return 0;
}



/*
 * memory requirement: 5 ring elements
 */

int encrypt_kem(
    const uint16_t  *m,     /* input binary message */
    const uint16_t  *h,     /* input public key */
          uint16_t  *c,     /* output ciphertext */
          uint16_t  *buf,
    const PARAM_SET *param)
{
    if (check_m(m, param->N) == -1 )
```

```
    {
        printf("error message\n");
        return -1;
    }
    uint16_t    i;
    uint16_t    *r, *t, *localbuf;


    r           = buf;
    t           = r + param->padN;
    /* three ring elements for karatsuba */
    localbuf    = t + param->padN;

    trinary_poly_gen(r, param->N, param->d);

    ntru_ring_mult_coefficients(r, h, param, localbuf, t);

    for (i=0;i<param->N;i++)
        c[i] = (t[i]*param->p + m[i]) & (param->q-1);

    memset(buf, 0, sizeof(uint16_t)*param->padN*5);
    return 0;
}

/*
 * lift the message back to a trinary polynomial
 * by mod q then mod p
 */
static void
lift_msg(
    uint16_t    *m,
    PARAM_SET   *param)
{
    uint16_t    i;
    int         tmp;
    for (i=0;i<param->N;i++)
    {
        tmp = m[i] % param->q;
```

```
        if (tmp>param->q/2)
            tmp -= param->q;

        tmp = tmp % param->p;
        if (tmp == 2)
            tmp = -1;
        if (tmp == -2)
            tmp = 1;
        m[i] = tmp;
    }


    return;
}


/*
 * memory requirement: 4 ring elements
 */
int decrypt_kem(
    uint16_t    *m,     /* output trinary message */
    uint16_t    *F,     /* input public key */
    uint16_t    *c,     /* input ciphertext */
    uint16_t    *buf,
    PARAM_SET   *param)
{

    uint16_t    *f, *localbuf, i;

    f           = buf;
    localbuf    = f + param->padN;

    for (i=0;i<param->N;i++)
        f[i] = F[i]*param->p;
    f[0]++;

    /* compute e = c * f */
    ntru_ring_mult_coefficients(c, f, param, localbuf, m);
```

```
    /* recover m = e mod p */
    lift_msg(m, param);

    memset(buf, 0, sizeof(uint16_t)*param->padN*4);
    return 0;
}



/*
 * check if a message length is valid for ntruencrypt-cca
 * then convert the message into a binary polynomial and
 * pad the message with a random binary string p
 */
int
pad_msg(
          uint16_t  *m,     /* output message */
    const char      *msg,   /* input message string */
    const size_t    msg_len,/* input length of the message */
    const PARAM_SET *param)
{
    if (msg_len > param->max_msg_len)
    {
        printf("error: message too long");
        return -1;
    }
    uint16_t    *pad;
    uint16_t    i,j;
    char        tmp;
    memset(m, 0, sizeof(uint16_t)*param->N);

    /* generate the pad of a degree 167 trinary polynomial*/
    pad = m + param->N - 167;
    trinary_poly_gen(pad, 167, 56);

    /* convert the message length into coefficients */
    pad -= 8;
```

```
    tmp = msg_len;
    for(j=0;j<8;j++)
    {
        pad[j] = tmp & 1;
        tmp >>= 1;
    }
    /* form the message binary polynomial */
    for (i=0;i<msg_len;i++)
    {
        tmp = msg[i];
        for(j=0;j<8;j++)
        {
            m[i*8+j] = tmp & 1;
            tmp >>= 1;
        }
    }
    return 0;
}


/*
 * converting a binary polynomial into a char string
 * return the length of the message string
 */
int
recover_msg(
          char      *msg,   /* output message string */
    const uint16_t  *m,     /* input binary message */
    const PARAM_SET *param)
{
    char    tmp;
    int     msg_len;
    uint16_t i,j;
    msg_len = 0;

    for (j=0;j<8;j++)
    {
        msg_len += (m[param->N - 167 - 8 + j]<<j);
```

```
    }

    if (msg_len > param->max_msg_len)
    {
        printf("error: message too long");
        return -1;
    }

    for (i=0;i<msg_len;i++)
    {
        tmp = 0;
        for (j=0;j<8;j++)
        {
            tmp += (m[i*8+j]<<j);
        }
        msg[i] = tmp;
    }
    return msg_len;
}



/*
 * generate a balanced trinary r from msg and h
 * memory requirement: 2 * LENGTH_OF_HASH
 */
int
generate_r(
          uint16_t  *r,     /* output r */
    const uint16_t  *msg,   /* input binary message */
    const uint16_t  *h,     /* input public key */
          uint16_t  *buf,
    const PARAM_SET *param)
{
    uint16_t i;
    for (i=0;i<param->N;i++)
    {
        if (msg[i]!=0 && msg[i]!=1 && (msg[i]%param->q)!=param->q-1)
```

```
        {
            printf("invalid messages\n");
            return -1;
        }
    }
    unsigned char *seed = (unsigned char*) buf;
    memset(seed, 0, sizeof(unsigned char)* LENGTH_OF_HASH*2);

    /* hash message/public key into a string 'seed'*/
    crypto_hash_sha512(seed, (unsigned char*)msg, param->N*2);
    crypto_hash_sha512(seed+LENGTH_OF_HASH, (unsigned char*)h, param->N*2);


    /* use the seed to generate r */
    trinary_poly_gen_w_seed(r, param->N, param->d, seed, LENGTH_OF_HASH*2);

    memset(seed, 0, sizeof(unsigned char)* LENGTH_OF_HASH*2);

    return 0;
}



/*
 * input a message msg, output msg \trixor hash(rh)
 * memory requirements: LENGTH_OF_HASH + 1 ring element
 */
int
mask_m(
          uint16_t  *msg,   /* in/output binary message */
    const uint16_t  *rh,
          uint16_t  *buf,
    const PARAM_SET *param)
{
    unsigned char   *seed;
    uint16_t        *mask;
    uint16_t        i;
```

```
    memset(buf,          0,          sizeof(uint16_t)*param->padN          +
sizeof(char)*LENGTH_OF_HASH);
    seed  = (unsigned char*) buf;
    mask  = (uint16_t *) (seed + LENGTH_OF_HASH);

    crypto_hash_sha512(seed,      (unsigned      char*)      rh,      param-
>N*sizeof(uint16_t)/sizeof(unsigned char));

    rand_tri_poly_from_seed(mask, param->N, seed, LENGTH_OF_HASH);

    for (i=0;i<param->N;i++)
    {
        if (mask[i] == 65535)
            msg[i] --;
        else if (mask[i] == 1)
            msg[i] ++;

        if (msg[i] == 65534)
            msg[i] =  1;
        if (msg[i] == 2)
            msg[i] =  -1;
    }
    memset(buf,          0,          sizeof(uint16_t)*param->padN          +
sizeof(char)*LENGTH_OF_HASH);
    return 0;
}



/*
 * input a message msg, output msg \trixor hash(rh)
 * memory requirements: LENGTH_OF_HASH + 1 ring element
 */
static int
unmask_m(
          uint16_t  *msg,   /* in/output binary message */
    const uint16_t  *rh,
          uint16_t  *buf,
```

```
    const PARAM_SET *param)
{
    unsigned char   *seed;
    uint16_t        *mask;
    uint16_t        i;

    memset(buf,          0,          sizeof(uint16_t)*param->padN          +
sizeof(char)*LENGTH_OF_HASH);
    seed  = (unsigned char*) buf;
    mask  = (uint16_t *) (seed + LENGTH_OF_HASH);


    crypto_hash_sha512(seed,      (unsigned      char*)      rh,      param-
>N*sizeof(uint16_t)/sizeof(unsigned char));

    rand_tri_poly_from_seed(mask, param->N, seed, LENGTH_OF_HASH);

    for (i=0;i<param->N;i++)
    {
        if (mask[i] == 65535)
            msg[i] ++;
        else if (mask[i] == 1)
            msg[i] --;

        if (msg[i] == 65534)
            msg[i] =  1;
        if (msg[i] == 2)
            msg[i] =  -1;
    }
    memset(buf,          0,          sizeof(uint16_t)*param->padN          +
sizeof(char)*LENGTH_OF_HASH);
    return 0;
}
/*
 * CCA-2 secure encryption algorithm using NAEP
 * memory requirement: 6 ring elements
 */
void
```

```
encrypt_cca(
          uint16_t  *c,     /* output ciphertext */
    const char      *msg,   /* input message: a string of chars */
    const size_t    msg_len,/* input the length of the message */
    const uint16_t  *h,     /* input public key */
          uint16_t  *buf,
    const PARAM_SET *param)
{
    uint16_t    i;
    uint16_t    *r, *t, *m, *localbuf;

    m           = buf;
    r           = buf   + param->padN;
    t           = r     + param->padN;
    localbuf    = t     + param->padN;

    /* pad the message */
    if (pad_msg( m, msg, msg_len, param) == -1)
        return;

    /* generate r from the message */
    if (generate_r(r, m, h, localbuf,param) == -1)
        return;

    /* compute r*h */
    ntru_ring_mult_coefficients(r, h, param, localbuf, t);
    for (i=0;i<param->N;i++)
    {
        t[i] *= param->p;
        t[i] &= (param->q-1);
    }

    /* mask the message with hash(r*h) */
    mask_m (m, t, localbuf, param);


    for (i=0;i<param->N;i++)
        c[i] = (t[i] + m[i]) & (param->q-1);
```

```
    memset(buf,0, sizeof(uint16_t)*param->padN*6);

    return ;
}


/*
 * CCA-2 secure encryption algorithm using NAEP
 * return the length of the message
 * memory requirement: 7 ring elements
 */
int decrypt_cca(
          char      *msg,  /* output message: a string of chars */
    const uint16_t  *F,    /* input public key */
    const uint16_t  *h,    /* input public key */
    const uint16_t  *c,    /* input ciphertext */
          uint16_t  *buf,
    const PARAM_SET *param)
{
    uint16_t i, msg_len;
    uint16_t *f, *m, *t, *r, *t_rec, *localbuf;

    memset(buf, 0, sizeof(int16_t)*param->padN*8);

    f           = buf;
    m           = f     + param->padN;
    t           = m     + param->padN;
    r           = t     + param->padN;
    t_rec       = r     + param->padN;
    localbuf    = t_rec + param->padN;


    for (i=0;i<param->N;i++)
        f[i] = F[i]*param->p;
    f[0]++;
```

```
/* compute e = c * f */
ntru_ring_mult_coefficients(c, f, param, localbuf, m);

/* recover m = e mod p */
lift_msg(m, param);

/* recover r*h */
for (i=0;i<param->padN;i++)
    t[i] = (c[i] - m[i]) & (param->q-1);

/* unmask m with hash(r*h) */
unmask_m (m, t, localbuf, param);

/* recover r from hash(m) */
if (generate_r(r, m, h, localbuf,param) == -1)
{
    memset(buf,0, sizeof(uint16_t)*param->padN*7);
    return -1;
}

/* check if recovered r is correct */
ntru_ring_mult_coefficients(r, h, param, localbuf, t_rec);

for(i=0;i<param->N;i++)
{
    if (((param->p*t_rec[i] - t[i]) & (param->q-1)) !=0)
    {
        printf("error: \n");
        printf("r: \n");
        for (i=0;i<param->padN;i++)
            printf("%d, ", r[i]);
        printf("\n");
        printf("h: \n");
        for (i=0;i<param->padN;i++)
            printf("%d, ", h[i]);
        printf("\n");
        printf("t_rec: \n");
        for (i=0;i<param->padN;i++)
```

```
                printf("%d, ", t_rec[i]);
            printf("\n");
            printf("t: \n");
            for (i=0;i<param->padN;i++)
                printf("%d, ", t[i]);
            printf("\n");
            printf("c: \n");
            for (i=0;i<param->padN;i++)
                printf("%d, ", c[i]);
            printf("\n");

            memset(buf,0, sizeof(uint16_t)*param->padN*8);
            return -1;
        }
    }

    /* convert the message polynomial into char string */
    msg_len = recover_msg(msg, m, param);
    memset(buf,0, sizeof(uint16_t)*param->padN*8);
    return msg_len;
}
```

Фрагмент коду 2 – Алгоритм несиметричного шифрування NTRU-KEM