

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій
(повне найменування факультету)

Кафедра «Інформаційна безпека та наноелектроніка»
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему Дослідження постквантових криптографічних
(назва теми)

алгоритмів на решітках

Виконав: студент 6 курсу, групи БКз-813м

Спеціальності 125 Кібербезпека та захист
(код і найменування спеціальності)
інформації

Освітня програма (спеціалізація)
безпека інформаційних і комунікаційних систем
ЄРШОВ А.М.

(ПРИЗВИЩЕ та ініціали)

Керівник НЕЛАСА Г.В.

(ПРИЗВИЩЕ та ініціали)

Рецензент САМОЙЛИК С.С.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет Інформаційної безпеки та електронних комунікацій
Кафедра Інформаційна безпека та нанoeлектроніка
Ступінь вищої освіти магістр
Спеціальність 125 Кібербезпека та захист інформації
(код і найменування)
Освітня програма (спеціалізація) Безпека інформаційних і комунікаційних
(назва освітньої програми (спеціалізації))
систем

Завідувач кафедри ІБтаН

« » 2024 року

ЗАВДАННЯ

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ЄРШОВА Андрія Миколайовича.

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження постквантових криптографічних алгоритмів на решітках Research of post-quantum lattice-based cryptography algorithms

керівник проєкту (роботи) к.т.н., доцент, НЕЛАСА Ганна Вікторівна

(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «05» грудня 2024 року №507

2. Строк подання студентом проєкту (роботи)

3. Вихідні дані до проєкту (роботи) наукові статті та дослідження за темою постквантових алгоритмів; реалізації алгоритму LWE, архів Cryptology ePrint

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) теоретичні основи постквантових алгоритмів; дослідження постквантових алгоритмів на решітках; розробка рішень та створення підсумків; порівняння алгоритмів; висновки; список використаних джерел; додатки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Презентація PowerPoint (17 слайдів)

6. Консультанти розділів проекту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	НЕЛАСА Г.В., доцент кафедри ІБтаН	04.09.2024	10.12.2024
Нормконтроль	КОРОЛЬКОВ Р.Ю. доцент кафедри ІБтаН		16.12.2024

7. Дата видачі завдання « 04 » жовтня 2024 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Ознайомлення з завданням до кваліфікаційної роботи	07.10.2024 – 09.10.2024	Виконано
2	Пошук наукових статей та досліджень за темою	11.10.2024 – 18.10.2024	Виконано
3	Систематизація зібраної інформації	19.10.2024 – 21.10.2024	Виконано
4	Підготовка методик тестування алгоритмів	22.10.2024 – 28.10.2024	Виконано
5	Обробка та аналіз отриманих даних	29.10.2024 – 11.11.2024	Виконано
6	Підбиття підсумків роботи, формулювання висновків	12.11.2024 – 02.12.2024	Виконано
7	Створення презентаційних матеріалів	03.12.2024 – 10.12.2024	Виконано
8	Оформлення кваліфікаційної роботи	11.12.2024 – 15.12.2024	Виконано
9	Нормоконтроль	16.12.2024	Виконано
10	Перевірка на плагіат	23.12.2024	Виконано
11	Захист кваліфікаційної роботи	27.12.2024	

Студент

_____ (підпис)

Андрій ЄРШОВ
(Ім'я ПРИЗВИЩЕ)

Керівник проекту (роботи)

_____ (підпис)

Ганна НЕЛАСА
(Ім'я ПРИЗВИЩЕ)

АНОТАЦІЯ

Пояснювальна записка до магістерської роботи: 63 с., 33 рис., 2 дод., 20 джерел.

ПОСТКВАНТОВА КРИПТОГРАФІЯ, РЕШІТКИ, NTRU, LWE, КВАНТОВІ ОБЧИСЛЕННЯ, ШИФРУВАННЯ, PYTHON.

Об'єктом дослідження є постквантові алгоритми, їх теоретичні основи, практичне застосування, а також проблеми та перспективи розвитку.

Метою роботи є вивчення та аналіз постквантових алгоритмів, їх можливостей та обмежень, а також перспектив застосування в різних галузях.

Предметом дослідження є програмний комплекс для тестування алгоритмів генерації ключів, шифрування та розшифрування.

У магістерській роботі досліджено постквантові криптографічні алгоритми на основі решіток, зокрема сімейства NTRU та LWE.

Проведено експериментальний аналіз часу виконання операцій і показано, що навіть у спрощеній реалізації NTRU та LWE можуть забезпечувати достатньо швидку роботу за належного вибору параметрів. Окрім того, виконано огляд безпекових характеристик, порівняння алгоритмів та наведено рекомендації щодо подальшого вдосконалення для досягнення підвищеної стійкості проти квантових атак.

ABSTRACT

Explanatory note to the diploma project: 63 p, 33 fig., 2 app., 20 references.

POST-QUANTUM CRYPTOGRAPHY, LATTICES, NTRU, LWE,
QUANTUM COMPUTATIONS, ENCRYPTION, PYTHON

The object of the research is post-quantum algorithms, their theoretical foundations, practical applications, as well as problems and prospects for development.

The purpose of the work is to study and analyze post-quantum algorithms, their capabilities and limitations, as well as prospects for application in various industries.

The subject of the research is a software package for testing key generation, encryption, and decryption algorithms.

The master's thesis investigates post-quantum cryptographic algorithms based on lattices, in particular the NTRU and LWE families.

An experimental analysis of the execution time of operations is carried out and it is shown that even in a simplified implementation, NTRU and LWE can provide sufficiently fast operation with the appropriate choice of parameters. In addition, a review of security characteristics, a comparison of algorithms and recommendations for further improvement to achieve increased resistance against quantum attacks are provided.

ЗМІСТ

Перелік скорочень та позначень	7
Вступ	9
1 Теоретичні засади постквантових алгоритмів	11
1.1 Основні поняття та визначення	11
1.2 Переваги методу LWE	13
1.3 Постановка задачі	15
2 Математичні основи обчислень на решітках	16
2.1 Евклідові решітки: означення та властивості	16
2.2 Основні задачі на решітках	17
2.3 Висновки до розділу	20
3 Обґрунтування проєктних рішень	21
3.1 Загальний підхід до організації дослідження	21
3.2 Архітектура програмного комплексу	22
3.3 Опис основних компонентів	23
4 Порівняння криптосистем та методика оцінювання	30
4.1 Порівняння за різними критеріями	30
4.2 Аналіз результатів порівняння	31
4.3 Огляд варіантів: Ring-LWE, Module-LWE, NTRU Prime	36
4.4 Практичні висновки та рекомендації	38
Висновки	40
Перелік джерел посилання	42
Додаток А Текст програми	44
Додаток Б Слайди презентації	55

ПЕРЕЛІК СКОРОЧЕНЬ ТА ПОЗНАЧЕНЬ

NTRU (N-th Degree Truncated Polynomial Ring Units) – криптосистема на основі решіток, побудована на множенні поліномів.

LWE (Learning With Errors) - криптографічна задача “навчання з похибками”.

Ring-LWE – версія LWE, де вектори a, s, e “згорнуті” до поліномів у кільці.

Module-LWE – узагальнення Ring-LWE на випадок “матриць поліномів”.

SIS (Short Integer Solutions) – задача короткого цілочисельного розв’язку, еквівалентна деяким задачам на решітках.

SVP (Shortest Vector Problem) – задача пошуку найкоротшого ненульового вектора в решітці.

CVP (Closest Vector Problem) – задача пошуку вектора в решітці, який найближчий до заданого вектора $t \in \mathbb{R}^n$.

LLL (Lenstra–Lenstra–Lovász) – алгоритм редукції базису решітки, що може шукати відносно короткі вектори.

BKZ (Block Korkine–Zolotarev) – розширення алгоритму LLL.

NTT/FFT (Number Theoretic Transform / Fast Fourier Transform) – методи швидкого множення поліномів.

ZK (Zero-Knowledge) – протоколи з нульовим розголошенням.

FHE (Fully Homomorphic Encryption) – повністю гомоморфне шифрування.

NIST PQC – ініціатива Національного інституту стандартів і технологій (NIST) з відбору постквантових криптографічних стандартів.

KEM (Key Encapsulation Mechanism) – механізм обміну секретним ключем у постквантових алгоритмах.

ECC (Elliptic Curve Cryptography) – класичні криптосистеми на еліптичних кривих.

RSA (аббревіатура від прізвищ Rivest, Shamir та Adleman) – класичний алгоритм з відкритим ключем.

CVP, SVP, SIS, LWE – Загальний перелік важких задач на решітках.

JSON (JavaScript Object Notation) - текстовий формат обміну даними

ВСТУП

У сучасному світі криптографія відіграє центральну роль у забезпеченні безпеки даних під час передачі та зберігання інформації. Класичні криптосистеми, побудовані на складних обчислювальних задачах (таких як факторизація великих чисел чи задача дискретного логарифму), вважаються безпечними для існуючих комп'ютерів. Однак, із появою квантових обчислювальних пристроїв традиційні алгоритми шифрування можуть стати вразливими: квантові алгоритми Шора та Гровера показують, що знайдення приватних ключів може відбуватися у рази швидше на квантовому комп'ютері, ніж на класичному.

Звідси виникає гостра потреба у розробці постквантових криптографічних алгоритмів, які будуть стійкими до атак квантових комп'ютерів. Одним із найперспективніших напрямків постквантової криптографії є криптосистеми, побудовані на ґратках. Такі алгоритми вже розглядаються як кандидати для стандарту наступного покоління (зокрема, в рамках ініціативи NIST з відбору постквантових стандартів).

Значна частина криптографічних алгоритмів на ґратках базується на проблемах, що пов'язані з багатовимірними векторами та їх проєкціями. Існує кілька відомих криптосистем на ґратках, які вже пройшли ретельне тестування та аналіз. Наприклад, алгоритми LWE та NTRU є одними з найпоширеніших і добре досліджених. Вони забезпечують високу безпеку і ефективність, що робить їх перспективними кандидатами для стандартизації.

Крім того, у постквантовій криптографії розвиваються нові гібридні методи, які об'єднують переваги різних алгоритмів для досягнення найвищої стійкості до атак. Наприклад, об'єднання ґраткових методів з класичними алгоритмами шифрування дозволяє створювати ще більш захищені системи.

У сучасних умовах важливо також проводити практичні дослідження та тестування криптосистем на ґратках. Це дозволяє виявити можливі вразливості та забезпечити високу надійність і ефективність алгоритмів у

реальних умовах. Актуальність таких досліджень підтверджується зростанням кількості квантових обчислювальних пристроїв та їх потенційним впливом на інформаційну безпеку.

Окрім стійкості до квантових обчислень, алгоритми на ґратках також мають інші переваги. Наприклад, вони забезпечують ефективне шифрування та дешифрування, мають можливість побудови гомоморфних шифрів, що дозволяє виконувати обчислення над зашифрованими даними без необхідності їх розшифровки. Це відкриває нові горизонти для безпечного оброблення даних у хмарних сервісах та інших сферах.

Таким чином, дослідження та розробка постквантових криптографічних алгоритмів на ґратках є надзвичайно актуальним напрямком, що потребує подальших наукових зусиль та інновацій. Це дозволить забезпечити надійний захист інформації у майбутньому світі, де квантові комп'ютери стануть реальністю.

1 ТЕОРЕТИЧНІ ЗАСАДИ ПОСТКВАНТОВИХ АЛГОРИТМІВ

1.1 Основні поняття та визначення

Решітка (Lattice) – у двовимірному випадку визначається двома лінійно незалежними векторами b_1 і b_2 у $\mathbb{R}^2$. У загальному випадку в n -вимірному просторі $\mathbb{R}^n$ решітка визначається базисом $B = \{b_1, b_2, \dots, b_n\}$. Формально:

$$\Lambda(B) = \left\{ \sum_{i=1}^n z_i b_i \mid z_i \in \mathbb{Z} \right\}$$

Задачі, пов'язані з решітками – задача пошуку найкоротшого вектора (SVP), задача пошуку найближчого вектора (CVP), задача пошуку рішення в решітці (SIS), задача навчання з похибками (LWE) – лежать в основі багатьох сучасних постквантових криптосистем. Їх складність (при належному виборі параметрів) вважається достатньо великою навіть для квантових обчислень.

Криптосистема NTRU – це одна з перших практичних криптосистем на решітках, розроблена ще в 1996 році (Hoffstein, Pipher, Silverman). Вона базується на багатовимірній поліноміальній решітці та вважається стійкою до відомих атак за умови правильно вибраних параметрів. Перевагою NTRU є відносно висока швидкодія операцій шифрування/розшифрування, а недоліком є розмір ключів, який зазвичай значно більший, ніж у класичних схемах RSA та ECC. Попри те, у контексті постквантової криптографії розмір ключів NTRU розглядається як прийнятний компроміс із огляду на безпечність.

Алгоритм LLL (Lenstra–Lenstra–Lovász) – це поліноміальний алгоритм редукції решіток, розроблений братами Ленстра та Ловасом у 1982 році. Він знаходить відносно “короткі” вектори в решітці та приводить базис до “зменшеної” (редукованої) форми. LLL є одним із центральних алгоритмів у криптоаналізі схем на решітках, оскільки дає змогу спростити або повністю

зламати певні варіанти криптосистем за недостатньо великих параметрів. Водночас він є важливим інструментом і для самих криптографічних застосувань, наприклад, у схемах цифрових підписів або у протоколах з доведенням без розголошення.

Zero-Knowledge-протоколи – це клас криптографічних протоколів, що дозволяють одній стороні (доказувачу) переконати іншу сторону (верифікатора) у справедливості певного твердження без розкриття будь-якої додаткової інформації. У контексті решіток такі протоколи зазвичай будуються на складності пошуку короткого вектора чи найближчого вектора у вирішуваній задачі. Зокрема, LWE слугує базисом для багатьох схем Zero-Knowledge, оскільки вважається еквівалентним певним важким задачам на решітках.

Learning With Errors (LWE) вперше було запропоновано Одедом Регевом (Oded Regev) у 2005 році [1]. Ідея полягає в тому, що маючи набір лінійних рівнянь із невеликою випадковою “похибкою” (errors), відновити вихідний секрет (наприклад, вектор s) є важкою задачею при належному виборі параметрів. LWE має декілька важливих властивостей:

1. Редукція до задач на решітках: Регев показав, що LWE-завдання можна звести до задачі на короткий вектор (SIS) у певній ґратці. Фактично доведена (за умови випадковості і коректного вибору розподілів) заснованість LWE на гіпотезі про складність задачі SIS чи Smallest Eigenvalue Problem.

2. Придатність для побудови різноманітних криптосистем: на базі LWE можна будувати схеми шифрування, цифрового підпису, ідентифікації, а також системи гомоморфного шифрування та Zero-Knowledge протоколи.

3. Стійкість до квантових обчислень: вважається, що LWE-параметри можна підібрати таким чином, щоб навіть квантові алгоритми не мали поліноміального прискорення для розв’язання задачі.

Порівняно з класичними криптосистемами на основі факторизації чи дискретного логарифму, LWE виграє у тому, що забезпечує вищий рівень безпеки проти атак квантовими алгоритмами. Гнучкий вибір параметрів

дозволяє “регулювати” безпеку та розмір ключів. Його простота та універсальність дозволяє будувати різні схеми (шифрування, підпису тощо) на одній базовій задачі.

Натомість недоліки LWE-рішень здебільшого пов’язані з розмірами ключів і шифротекстів, що можуть бути досить великими у порівнянні з традиційними підходами (RSA, ECC). Проте, з огляду на постквантову стійкість, це сприймається як прийнятний компроміс.

Проаналізувавши опубліковані роботи з ресурсу eprint.iacr.org [2-8] , у яких досліджується або застосовується LWE, можна зробити загальний висновок: LWE та його різноманітні похідні (Ring-LWE, Module-LWE тощо) залишаються одними з найбільш досліджуваних та перспективних напрямів постквантової криптографії. Покращуються як теоретичні аспекти (редукції, нові схеми), так і прикладні аспекти (ефективність реалізацій, застосування в гомоморфному шифруванні, захищених обчисленнях, Zero-Knowledge протоколах).

1.2 Переваги методу LWE

Безпека методу LWE ґрунтується на міцно обґрунтованих математичних принципах. Це включає складність задач на решітках та додавання помилок, що робить LWE надійним криптографічним інструментом.

LWE пропонує високий рівень безпеки, стійкість до квантових атак, гнучкість у налаштуванні та різноманітність криптографічних протоколів, що робить його одним з найважливіших і найперспективніших інструментів у постквантовій криптографії.

Завдяки своїй структурі, LWE дозволяє створювати криптосистеми з доказовою безпекою, що означає, що їх стійкість базується на чітко визначених математичних задачах.

Підсумовуючи вищезначене, можна виділити кілька ключових моментів, чому LWE розглядають як один із “кращих” кандидатів для постквантової криптографії:

- задачі на решітках, такі як SIS і SVP, є важливими компонентами сучасної криптографії та забезпечують високу стійкість до атак, включаючи атаки з використанням квантових комп'ютерів;
- на основі LWE будуються шифрування, підписи, ідентифікаційні протоколи, Zero-Knowledge, гомоморфне шифрування тощо;
- хоча необхідно обирати розміри, розподіл похибки і т. д., існує низка стандартних рекомендацій та емпіричних досліджень, які спрощують цей процес;
- активне дослідження та підтримка в академічній спільноті. Велика кількість робіт присвячена саме LWE, що дозволяє швидко виявляти та усувати потенційні вразливості або покращувати ефективність;
- стійкість до більшості відомих підходів криптоаналізу. Незважаючи на активні намагання дослідників знайти квантові чи класичні алгоритми швидкого розв'язання задачі LWE, наразі немає відомих поліноміальних алгоритмів.

Таким чином, LWE залишається одним із “стовпів” сучасної постквантової криптографії й закономірно розглядається як перспективна основа для побудови безпечних криптографічних систем. У майбутньому дослідження та розробка алгоритмів на основі LWE будуть продовжувати грати важливу роль у розвитку постквантової криптографії, забезпечуючи надійний захист інформаційних систем від потенційних загроз. Постійні інновації та вдосконалення у цій галузі дозволять зберегти безпеку даних в умовах швидкого технологічного прогресу.

1.3 Постановка задачі

Основною метою даної дипломної роботи є детальне вивчення, аналіз і практичне порівняння постквантових криптографічних алгоритмів на основі решіток. Для досягнення цієї мети передбачається виконання ряду завдань:

- вивчення математичних основ криптографії на решітках, включаючи означення та основні властивості евклідових решіток, задачі на решітках (SVP, CVP, SIS, LWE), алгоритм LLL як основний інструмент редукції базисів;
- аналіз NTRU, а також інших відомих криптосистем на решітках (особливо тих, які розглядаються у контексті постквантових стандартів);
- огляд протоколів Zero-Knowledge (ZK), що будуються на решітках, зокрема, важливість задачі LWE;
- перегляд останніх публікацій у галузі LWE, визначення, в яких напрямках застосовується та які переваги має LWE;
- проектування архітектури експериментальної платформи для проведення порівняльних експериментів із різними алгоритмами на решітках (NTRU та LWE);
- проведення експериментального порівняння швидкодії різних алгоритмів та аналіз результатів;
- підготовка висновків щодо практичного застосування досліджуваних алгоритмів та перспектив їхнього розвитку.

2 МАТЕМАТИЧНІ ОСНОВИ ОБЧИСЛЕНЬ НА РЕШІТКАХ

2.1 Евклідові решітки: означення та властивості

Давньогрецький математик Евклід, який жив приблизно в 300 року до н.е. написав свою знамениту працю "Начала" (або "Елементи"), що є однією з найвпливовіших книг в історії математики. У цій праці він описав основи геометрії та арифметики, включаючи поняття, які стали основою для розвитку Евклідових решіток

Евклідові решітки – це математичні структури, які представляють собою множину всіх лінійних комбінацій з цілочисельними коефіцієнтами базисних векторів у n -вимірному евклідовому просторі. Вони мають широке застосування в теорії чисел, криптографії та теорії інформації.

Евклідова решітка (англ. lattice) в n -вимірному просторі $\mathbb{R}^n$ визначається набором лінійно незалежних векторів (базисом) $B = \{b_1, \dots, b_n\}$. Кожен вектор b_i має розмірність n . Решітка $\Lambda(B)$ задається як множина всіх цілочисельних лінійних комбінацій базисних векторів:

$$\Lambda(B) = \left\{ \sum_{i=1}^n z_i b_i \mid z_i \in \mathbb{Z} \right\}$$

Сутність задач, пов'язаних із решітками, полягає у тому, що базис може бути “хорошим” або “поганим” із точки зору пошуку коротких чи найближчих векторів. Якщо базис має дуже короткі і майже ортогональні вектори, працювати з ним легше. Але для складних криптографічних задач зазвичай використовується “поганий” базис, де вектори довгі і сильно корельовані.

2.2 Основні задачі на решітках

Задачі на решітках (lattice problems) є фундаментальними в області постквантової криптографії. Вони зводяться до задач, що використовують структури решіток для створення стійких до квантових комп'ютерів алгоритмів.

Задача пошуку найкоротшого вектора (SVP) – для заданого базису B і решітки $\Lambda(B)$ потрібно знайти вектор $v \in \Lambda(B) \setminus \{0\}$ такий, що його норма (наприклад, Евклідова норма $\|v\|$) є мінімальною серед усіх ненульових векторів решітки. Задача належить до класу NP-повних (уточнення: SVP у різних метриках має деякі варіації складності, проте в Евклідовій метриці це широко визнана складна задача). Більшість методів розв'язання (навіть найбільш передових) мають експоненційну складність у розмірі решітки.

Задача найближчого вектора (CVP) – маючи решітку $\Lambda(B)$ і довільний вектор $t \in \mathbb{R}^n$, потрібно знайти такий вектор $v \in \Lambda(B)$, що $\|t-v\|$ буде мінімальною. CVP теж є NP-важкою задачею. Часто застосовується в аналізі стійкості схем, оскільки одна з атак може зводитися до пошуку “наближеного” вектора в решітці, сформованій з відкритих даних.

Задача SIS – потрібно знайти “короткий” розв'язок $x \in \mathbb{Z}^m$ (з малою нормою $\|x\|$), що задовольняє рівнянню:

$$Ax \equiv 0 \pmod{q},$$

де A – матриця розміру $n \times m$ з елементами з $\mathbb{Z}_q$, а q – деяке ціле (зазвичай просте).

SIS лежить в основі багатьох криптографічних конструкцій, особливо підписів і схем ідентифікації.

Задача LWE – для заданого (але прихованого) вектора $s \in \mathbb{Z}_q^n$ даються “приклади” лінійних рівнянь з шумовою похибкою:

$$a_i \cdot s + e_i \equiv b_i \pmod{q},$$

де a_i, b_i відомі, а e_i – невеликий випадковий “шум” із певного розподілу (зазвичай дискретного гауссового).

Потрібно за набором пар (a_i, b_i) відновити s . Вважається, що при належних параметрах задача є важкою навіть для квантових комп’ютерів. Регев (O. Regev) довів [1], що LWE зводиться до задач на решітках, зокрема SIS та варіантів задачі SVP, у випадку “правильної” генерації a_i .

Редуція базисів (алгоритм LLL) – це поліноміальний алгоритм, який приймає базис решітки B і повертає “редукований” базис B' із коротшими векторами, котрі водночас зберігають цілісну структуру решітки. Основний підхід полягає у послідовному застосуванні процедури Грама–Шмідта з регулярними “короткими” замінами базисних векторів. Критерій “редукованості” визначається певним параметром $\delta (1/4 < \delta < 1)$. LLL дає змогу знайти короткий вектор у решітці (не гарантовано найкоротший, але достатньо короткий), і цим іноді “ламає” криптосистему з недостатньо великими параметрами.

BKZ [9] – це більш потужна (але складніша й повільніша) альтернатива LLL, що розглядає “блоки” векторів і намагається покращити їхню ортогональність та мінімізувати довжину векторів у цих блоках. Цей алгоритм дозволяє досягати кращих (коротших) векторів, ніж LLL, проте має більшу обчислювальну складність. Для більшої точності можуть використовуватися розширені варіанти LLL або BKZ з більшими, хоча і субекспоненційними, витратами обчислень.

Криптосистеми на основі решіток поділяють за кількома критеріями: тип задачі (SIS, LWE, NTRU), тип застосування (шифрування, підпис, аутентифікація, гомоморфне шифрування тощо), представлення решітки (звичайні, кільцеві, модульні) тощо. Нижче наведено короткий огляд основних алгоритмів.

Криптосистема NTRU – запропонована у 1996 році (Hoffstein, Pipher, Silverman) [10]. Використовує поліноміальні кільця $\mathbb{Z}[x]/(x^n - 1)$ та специфічну структуру цих решіток для шифрування/розшифрування. Швидкий (особливо на стадії шифрування й розшифрування), стійкий до відомих атак за умови правильного вибору параметрів. Недолік: відносно великі розміри публічних ключів.

Криптосистема LWE – сформульована Оdedом Регевом (2005) [1], вважається базовим прикладом криптосистеми на решітках.

Ring-LWE – версія, де вектори a та s “упаковані” як поліноми в кільці

$$\mathbb{Z}_q[x]/(\Phi(x))$$

$$\Phi(x) = x^n + 1$$

Module-LWE – узагальнення Ring-LWE, де маємо “модулі” (матриці) замість окремих поліномів.

Застосування сімейства LWE – шифрування (як гомоморфне, так і звичайне), підписи, Zero-Knowledge протоколи, системи обміну ключами тощо.

Гомоморфне шифрування (Gentry FHE) [11] – перша повністю гомоморфна схема, що теж базується на решітках і застосовує ідеї перезавантаження.

Dilithium [12] – побудована на Module-LWE/SIS, є кандидатом NIST (3 раунд відбору постквантових алгоритмів).

FALCON [13] – використовує NTRU-решітки, ефективна, але складніша в реалізації (особливо через необхідність коректної гауссової вибірки).

Криптоаналіз схем на решітках переважно базується на редукції базисів (LLL, BKZ, тощо) та пошуку коротких/найближчих векторів. Практичні атаки на невеликі розміри (параметри) доводять, що системи можуть зламуватися за допомогою потужних оптимізованих реалізацій BKZ. Однак параметри, рекомендовані NIST та іншими дослідниками, є досить великими, аби такі атаки стали непрактичними.

2.3 Висновки до розділу

Решітки забезпечують складні базові задачі (SVP, CVP, SIS, LWE), які є твердими кандидатами на постквантову криптографію. Алгоритми редукції (LLL, BKZ) відіграють критичну роль: вони одночасно і дають змогу будувати, і дають змогу ламати криптосистеми.

Різні схеми на решітках (NTRU, LWE-сімейство, гомоморфні схеми) використовують різні аспекти цієї математики для забезпечення безпеки та ефективності.

Задачі SIS та LWE слугують фундаментом більшості сучасних розробок у галузі постквантових алгоритмів; крім шифрування, вони охоплюють підписи та ідентифікацію.

3 ОБГРУНТУВАННЯ ПРОЄКТНИХ РІШЕНЬ

У цьому розділі розглянуто практичні аспекти розробки експериментальної платформи (або тестового стенду) для дослідження різних криптографічних алгоритмів на решітках. Основна мета – описати архітектуру та основні компоненти проєкту, необхідні для реалізації, тестування і порівняння криптосистем на базі NTRU, LWE та інших.

3.1 Загальний підхід до організації дослідження

В якості середовища для реалізації криптографічних алгоритмів на решітках обрано Python 3 [14] та бібліотеки [15] [16] з реалізаціями алгоритмів NTRU та LWE.

Python забезпечує автоматичну підтримку довгих цілих чисел. Це суттєво спрощує криптографічні обчислення, оскільки не потрібно окремих бібліотек для “BigInt”.

Існують стандартні модулі json, time, random, secrets для реалізації основних операцій (запис/зчитування ключів, вимірювання часу, генерація випадкових чисел).

Структура проєкту мінімалістична: два основні скрипти (один — для генерації ключів, інший — для тестування) і два файли ключів (JSON).

Алгоритм роботи:

- крок 1: запуск сервера Node.js;
- крок 2: відкриття у ВЕБ-переглядачі сторінки із тестовим стендом;
- крок 3: проведення серії тестів;
- крок 4: аналіз отриманих даних.

План експериментів:

- запуск серій тестів (для NTRU, LWE) з різними параметрами;
- обробка статистичних даних;
- отримання висновків.

3.2 Архітектура програмного комплексу

Комплекс складається трьох складових: сервер Node.js, Python-скрипти тестування криптосистем, JavaScript-бібліотека для візуалізації отриманих результатів.

```
pqc-test/
├── html                # Пака з шаблонами сторінок
│   └── index.html      #
├── lwe                 # Бібліотека LWE
│   └── minimalist_lwe.py #
├── ntru                # Бібліотека NTRU
│   ├── fracModulo.py   #
│   ├── ntru.py         #
│   └── poly.py         #
├── compute-lwe-dec.py  # Python-скрипт тесту LWE (дешифр.)
├── compute-lwe-enc.py  # Python-скрипт тесту LWE (шифр.)
├── compute-ntru-dec.py # Python-скрипт тесту NTRU (дешифр.)
├── compute-ntru-enc.py # Python-скрипт тесту NTRU (шифр.)
└── server.js          # Node JS сервер
```

Рисунок. 3.1 – Архітектура програмного комплексу

Можливі обмеження та виклики:

- швидкодія – множення поліномів має складність $O(n^2)$; для дуже великих n (кілька сотень або тисяч) у Python це може бути повільно;
- оптимізація пам'яті – великі матриці (для LWE) можуть споживати значні обсяги пам'яті.



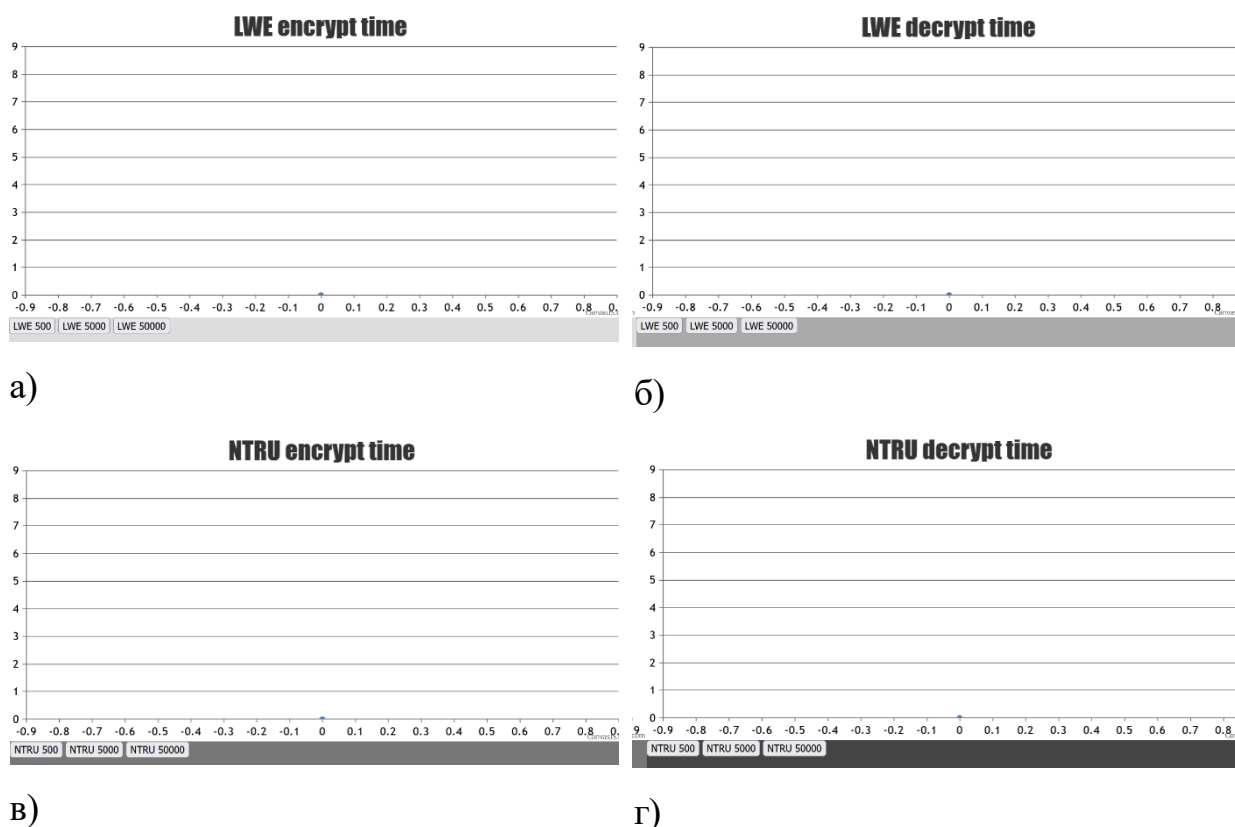
Рисунок 3.2 – Схематична діаграма програмного комплексу

3.3 Опис основних компонентів

WEB-сторінка (представлена на Рисунку 3.3) містить стенд тестування криптосистем.

Візуальна частина програми представлена у вигляді WEB-сторінки, що поділена на 4 області. Дві області для алгоритму LWE (шифрація/дешифрація), та дві області для алгоритму NTRU відповідно.

В цих областях розташовані графіки для візуалізації даних, що будуть отримані в ході проведення експериментальних робіт. Ці графіки реалізовано за допомогою JavaScript-бібліотеки CanvasJS [17]. Таке рішення допомагає легко збільшити кількість візуалізацій тестових даних, та дозволяє переглядати результати вимірювань в реальному часі. Також дуже зручним моментом є можливість масштабування графічної інформації без втрати презентативності.



а – графік швидкості шифрування (криптосистема LWE), б - графік швидкості дешифрування (криптосистема LWE), в – графік швидкості шифрування (криптосистема NTRU), г - графік швидкості дешифрування (криптосистема NTRU).

Рисунок 3.3 – Стенд тестування криптосистем

Графіки швидкості представлені на Рисунку 3.3 демонструють яку величину часу (в мс) зафіксовано в кожний окремий тест. Тобто, на осі абсцис відкладається кількість дослідів, а на осі ординат – величина часу, затраченого криптосистемою на операцію шифрування або дешифрування (в залежності від того, який саме тест проводиться).

Кожна область містить кнопки для запуску тестування відповідного алгоритму, та графік отриманих результатів. Кнопок запуску – три: з різною кількістю циклів (500, 5000 та 50000).

Після натискання на будь-яку кнопку виконується блокування всіх кнопок на час виконання тестових розрахунків (поки не прийде відповідь від сервера з результатом). Після отримання відповіді – блокування кнопок знімається, а тестовий стенд готовий до наступного тестування (Рисунок 3.4).

```
var disableButtons = function () {
    //buttons_div.hide();
    $('button').prop('disabled', true);
}
var enableButtons = function () {
    //buttons_div.show();
    $('button').prop('disabled', false);
}
```

Рисунок 3.4 – Активація / деактивація кнопок тестування

```
const path = require('path');
const express = require('express');
const { spawn } = require('child_process');

const app = express();
const port = 3000;

function runPythonScript(scriptPath, args, callback) {

app.get('/', function(req, res) {
    res.sendFile(path.join(__dirname, '/html/index.html'));
});
```

Рисунок 3.5 – Сервер Node.js

За допомогою кнопок виконуються GET-запити (з різними параметрами для тестування) що надсилаються на сервер Node.js (Рисунок 3.5). Зручним моментом є реалізація сервера за допомогою Express, що є програмним каркасом для розробки серверної частини вебзастосунків для Node.js. Таке рішення дозволяє швидко розробляти інфраструктуру програмного комплексу та допомагає мінімізувати помилки в ході проєктування серверної частини проєкту в цілому.

```

<!DOCTYPE HTML>
<html>
  <head>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
    <script>
      window.onload = function () {
    </script>
    <style>
  </head>
  <body>
    <div id="div1">
      <div id="chartContainer" style="height: 90%; width:100%;"></div>
      <div id="buttons_lwe_enc">
        <button type="button" name="lwe-enc" id="test_lwe_enc_1">LWE 500</button>
        <button type="button" name="lwe-enc" id="test_lwe_enc_2">LWE 5000</button>
        <button type="button" name="lwe-enc" id="test_lwe_enc_3">LWE 50000</button>
        <!--button type="button" name="lwe-enc" id="lwe_enc_reset">Reset</button-->
      </div>
    </div>
    <div id="div2">
    </div>
    <div id="div3">
    </div>
    <div id="div4">
    </div>
    <script src="https://cdn.canvasjs.com/canvasjs.min.js"></script>
  </body>
</html>

```

Рисунок 3.6 – HTML-документ сторінки тестового стенду

GET-запити від сторінки тестового стенду обробляються частинами сервера, які запускають відповідні скрипти Python (Рисунок 3.9). Це ядро програмного комплексу, яке складається з програмних реалізацій криптосистем, та допоміжних скриптів для їх використання.

```

before_decrypt_time = time.time()
for _ in range(iterations):
    NTRU_decrypted_message = Bob.decrypt(NTRU_encrypted_msg)
after_decrypt_time = time.time()
test_Time = after_decrypt_time - before_decrypt_time

```

Рисунок 3.7 – Вимірювання затраченого часу

Для отримання статистичних даних виконується вимірювання часу, затраченого на задану кількість циклів. Для цього використовується стандартна функція `time()` (Рисунок 3.7).

Після отримання результатів обчислень – вони надсилаються як відповідь на попередній запит від сторінки стенду (Рисунок 3.3).

```

app.get('/', function(req, res) {
  res.sendFile(path.join(__dirname, '/html/index.html'));
});

app.get('/pgc-test/:number', (req, res) => {

app.get('/lwe-enc/:number', (req, res) => {

app.get('/lwe-dec/:number', (req, res) => {
  const number = req.params.number;
  runPythonScript('compute-lwe-dec.py', [number], (err, result) => {
    if (err) {
      res.status(500).send(err);
    } else {
      res.send(`${result}`);
    }
  });
});

app.get('/ntru-enc/:number', (req, res) => {

app.get('/ntru-dec/:number', (req, res) => {

app.listen(port, () => {
  console.log(`Example app listening on port ${port}`)
})

```

Рисунок 3.8 – Частина серверу Node.js що обробляє GET-запити

```

import sys
import json
from lwe.minimalist_lwe import MinimalistLWE

from random import randint
import time

if __name__ == '__main__':
  iterations = int(sys.argv[1]) # Takes number from command line argument

  msg = randint(1000, 2000)
  test_Time = 0

  MinimalistLWE = MinimalistLWE(97, 100)
  MinimalistLWE.gen_publicKey()
  sk = MinimalistLWE.get_privateKey()

  before_encrypt_time = time.time()
  for _ in range(iterations):
    encrypted_message = MinimalistLWE.min_encrypt(msg)
  after_encrypt_time = time.time()
  test_Time = after_encrypt_time - before_encrypt_time

  print(test_Time)
  sys.stdout.flush()

```

Рисунок 3.9 – Python-скрипт використання бібліотеки MinimalistLWE

```

// CHART 1 -----
var dps_1 = []; // dataPoints
var chart_1 = new CanvasJS.Chart("chartContainer", {
    title :{ text: "LWE encrypt time" },
    data: [{
        type: "line",
        dataPoints: dps_1
    }]
});

var xVal = 0;
var yVal = 100;

var updateChart1 = function (yVal_local) {
    dps_1.push({
        x: xVal,
        y: yVal_local
    });
    xVal++;

    if (dps_1.length > dataLength) { dps_1.shift(); }

    chart_1.render();
};
// init chart 1
updateChart1(0);

```

Рисунок 3.10 – Використання бібліотеки CanvasJS для графіків

Окрім допоміжних функцій, таких як перемикання режиму доступу до тестових кнопок (для запобігання занадто частого запуску тестових задач), був також написаний додатковий код на JavaScript для забезпечення можливості переглядати 4 графіка одночасно (Рисунок 3.10).

Для спрощення кодової бази була використана jQuery [18] - популярна JavaScript-бібліотека з відкритим кодом. Використання подібних рішень значно пришвидшує процес створення функціоналу для візуальної частини програмного комплексу, та допомагає уникнути проблем «кросс-браузерності» (явище, коли оди і той-самий код виконується різними веб-переглядачами по-різному, або з помилками).

```

var submitTest = function (button_name, iterations) {
    var button_obj = $('#' + button_name);
    if (button_obj.is(':disabled')) { return } else {
        disableButtons();
    }

    $.ajaxSetup({async: false});
    $.ajax({
        type: "GET",
        url: '/' + button_name + '/' + iterations,
        success: function (response) {
            console.log(button_name + ': ' + response);
            // UPDATE CHART - use 1000 for milliseconds in chart
            switch (button_name) {
                case 'lwe-enc':
                    updateChart1(response * 1000);
                    break;
                case 'lwe-dec':
                    updateChart2(response * 1000);
                    break;
                case 'ntru-enc':
                    updateChart3(response * 1000);
                    break;
                case 'ntru-dec':
                    updateChart4(response * 1000);
                    break;
                default:
                    console.log("WARNING: There is no button " + button_name);
            }

            setInterval(function() {
                enableButtons();
            }, 10);
        },
    });
};

```

Рисунок 3.11 – Створення запиту та оновлення графіків

Запропонована архітектура дозволяє централізовано реалізовувати, тестувати й порівнювати реалізації криптосистем NTRU, LWE (або інші алгоритми – для цього достатньо підключити необхідну бібліотеку).

Модуль тестування спрощує проведення великої кількості експериментів і надає змогу порівняти різні схеми з різними параметрами.

Підхід до візуалізації даних забезпечує можливість повторюваності експериментів і спрощує аналіз.

4 ПОРІВНЯННЯ КРИПТОСИСТЕМ ТА МЕТОДИКА ОЦІНЮВАННЯ

4.1 Порівняння за різними критеріями

Безпековий рівень залежить від теоретичної складності основної математичної задачі (SVP, SIS, LWE). Також враховується експериментальний статус атак методом редукції базисів (LLL, BKZ) та їхніх гібридних варіантів. Усі розглядувані схеми (NTRU, LWE) можуть бути налаштовані різними параметрами.

Розмір публічного/приватного ключів. Як правило, NTRU має менший (чи співставний) розмір публічного ключа, ніж звичайний LWE, але більший, ніж Ring-LWE та Module-LWE-аналоги. LWE-базована схема в класичному варіанті потребує матриць і векторів великої розмірності, тож має доволі масивні ключі.

Порівняння часу шифрування/розшифрування. NTRU давно відома високою швидкістю шифрування/розшифрування. При цьому генерація ключів може займати більше часу, особливо коли потрібно знайти обернений поліном. LWE (в загальному вигляді) іноді повільніша на етапі шифрування/розшифрування, але має зручну реалізацію, особливо в Ring-LWE, де використовується NTT (Number Theoretic Transform) для швидкого множення.

Складність реалізації. NTRU вимагає реалізації функцій інвертування поліномів у кільці $Z_q[x]/(x^n - 1)$.

LWE, Ring-LWE іноді реалізуються простіше, якщо є готові бібліотеки FFT/NTT (для швидкого множення) і дискретного гауссового семплінгу. Усі згадані схеми сьогодні мають відповідні реалізації, тож із точки зору розробника простіше взяти готові бібліотеки.

Практичне застосування. NTRU – одна з перших постквантових схем шифрування, включена до набору фіналістів у деяких проєктах, зокрема в рамках NIST PQC (варіанти NTRU-HPS, NTRU-HRSS).

LWE/Ring-LWE – основа для більшості схем із гомоморфним шифруванням, підписів (Dilithium), ключових обмінів (Kyber) тощо.

В контексті безпечності NTRU і Kyber (LWE) – серед найактивніших кандидатів.

4.2 Аналіз результатів порівняння

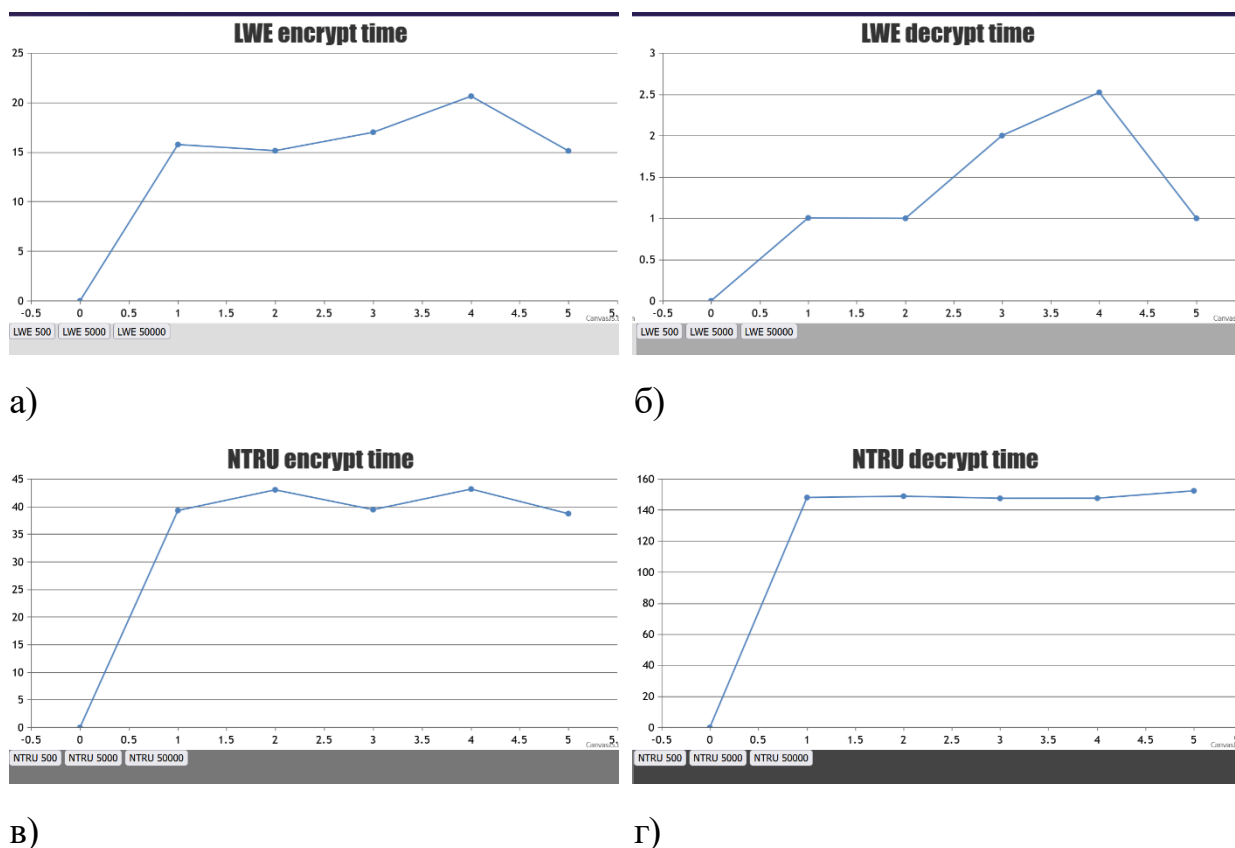
У традиційних реалізаціях LWE з матрицями A розміру $m \times n$ публічний ключ може бути досить великим (оскільки необхідно зберігати велику кількість елементів).

NTRU має коротший ключ, принаймні в порівнянні з криптосистемою LW у класичному варіанті”.

Ring-LWE або Module-LWE можуть скоротити розмір матриці за рахунок циклічної структури.

NTRU зазвичай швидко шифрує і розшифровує, але потребує додаткових кроків під час KeyGen (зокрема, пошук інвертованого полінома, що може збільшити час.

LWE (класичний) має досить швидке шифрування/розшифрування, але всі операції вимагають множення матриць, що може бути відносно повільним. При цьому сучасні FFT/NTT-оптимізації дозволяють пришвидшити Ring-LWE.

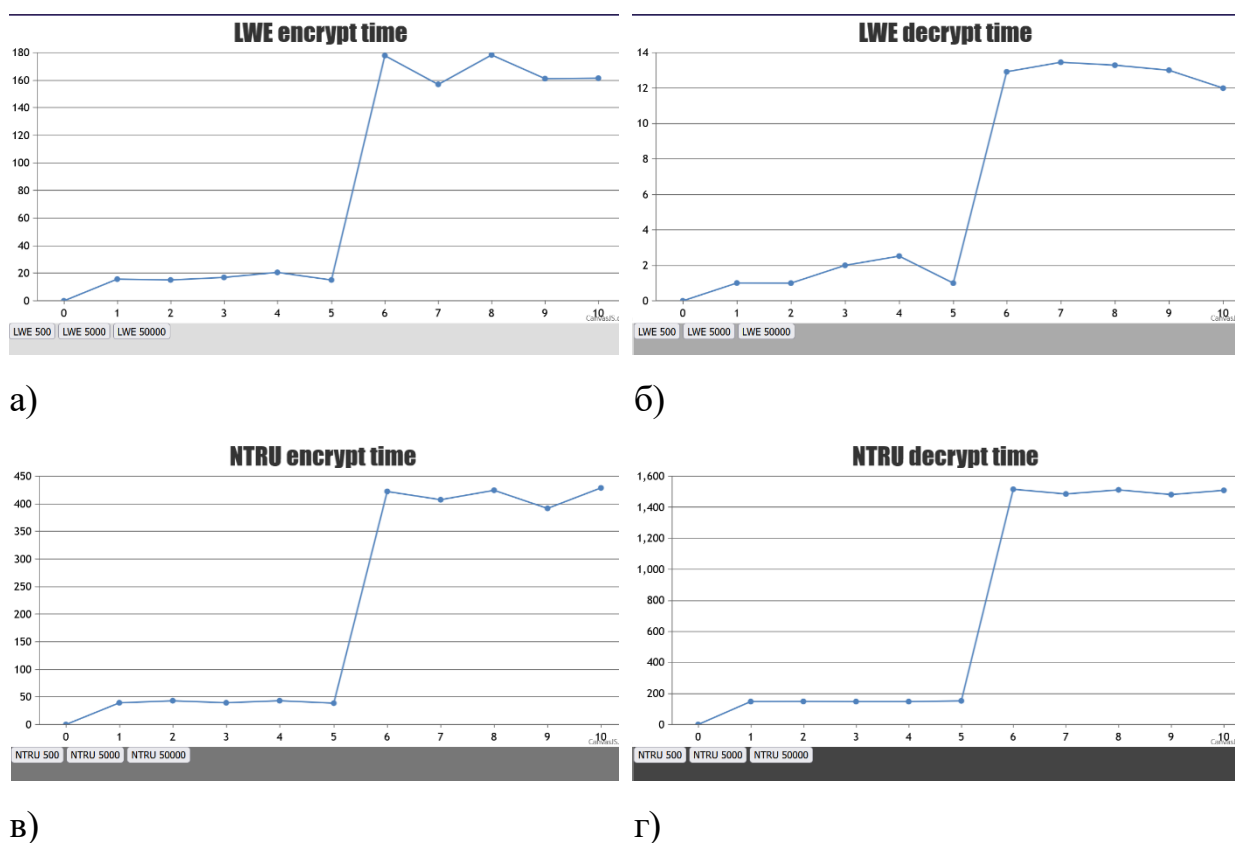


а – графік швидкості шифрування (криптосистема LWE – 500 циклів),
 б - графік швидкості дешифрування (криптосистема LWE – 500 циклів),
 в – графік швидкості шифрування (криптосистема NTRU – 500 циклів),
 г - графік швидкості дешифрування (криптосистема NTRU – 500 циклів).

Рисунок 4.1 – Тестування криптосистем LWE та NTRU (500 циклів)

Як видно з Рисунок 4.1, реалізація криптосистеми NTRU має час виконання операції дешифрування на два порядки більший, ніж та сама операція у криптосистемі LWE (150 мс проти 1-2,5 мс відповідно).

Час затрачений на операцію шифрування обидві криптосистеми мають приблизно однакового порядку (NTRU: 40-45 мс, LWE: 15-20 мс).

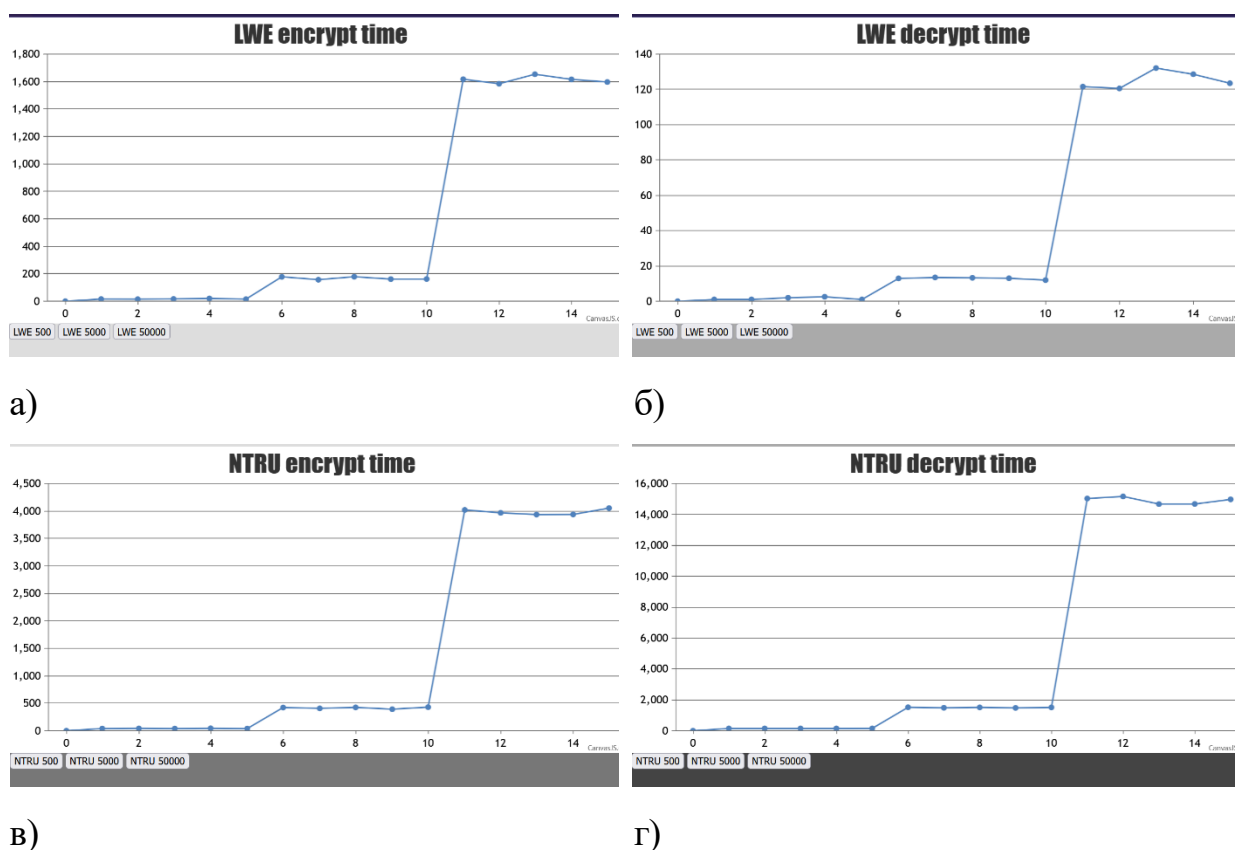


а – графік швидкості шифрування (криптосистема LWE – 5000 циклів),
 б - графік швидкості дешифрування (криптосистема LWE – 5000 циклів),
 в – графік швидкості шифрування (криптосистема NTRU – 5000 циклів),
 г - графік швидкості дешифрування (криптосистема NTRU – 5000 циклів).

Рисунок 4.2 – Тестування криптосистем LWE та NTRU (5000 циклів)

На Рисунку 4.2 ліва частина кожного з графіків містить результати попередніх вимірювань (500 циклів). В правій частині кожного графіка можна побачити результати другої серії вимірювань (5000 циклів).

Таке рішення дозволяє наочно спостерігати, залежність часу виконання алгоритму від кількості циклів в тестуванні.



а – графік швидкості шифрування (криптосистема LWE – 50000 циклів),
 б - графік швидкості дешифрування (криптосистема LWE – 50000 циклів),
 в – графік швидкості шифрування (криптосистема NTRU – 50000 циклів),
 г - графік швидкості дешифрування (криптосистема NTRU – 50000 циклів).

Рисунок 4.3 – Тестування криптосистем LWE та NTRU (50000 циклів)

На Рисунку 4.3 в правій частині кожного графіка зображено результати третьої серії вимірювань (50000 циклів). Легко побачити, що із зростанням кількості циклів, розкид значень затраченого часу зменшується. Це дає змогу зробити висновки щодо стабільності обраних бібліотек криптосистем. Зокрема, представлена реалізація криптосистеми NTRU продемонструвала більшу стабільність. В той час, як LWE показала досить великий розкид значень в першій серії вимірювань, але в наступних серіях випробувань цей розкид зменшився до значень зіставних з такими у криптосистеми NTRU.

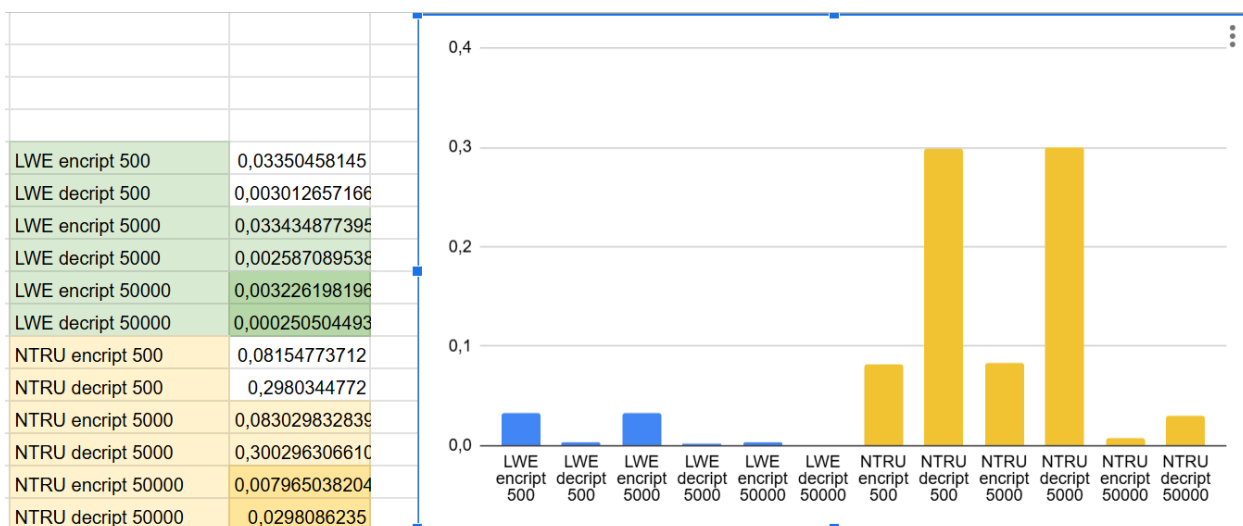


Рисунок 4.4 – Аналіз отриманих даних (500 – 50000 циклів)

Як видно з Рисунок 4.4 представлена реалізація алгоритму LWE значно виграє. Середня швидкість шифрування – не більше 0,033 мс. В той час, як у реалізації алгоритму NTRU цей показник – близько 0,3 мс (при кількості циклів 500 та 5000).

Результати тестів дешифрування показують ще більшу різницю. У алгоритму LWE - 0,003 мс, а у алгоритму NTRU – 0,3 мс (при кількості циклів 500 та 5000 відповідно).

Цікавими моментом є те, що при кількості ітерацій 50000 спостерігається значне зменшення середнього часу виконання операцій шифрування та дешифрування для кожного алгоритму. Цей феномен швидше за все пов'язаний з оптимізацією на рівні процесора. Тестування проводилось на ноутбучі з процесором AMD Ryzen 5 5600H (3.30 GHz).

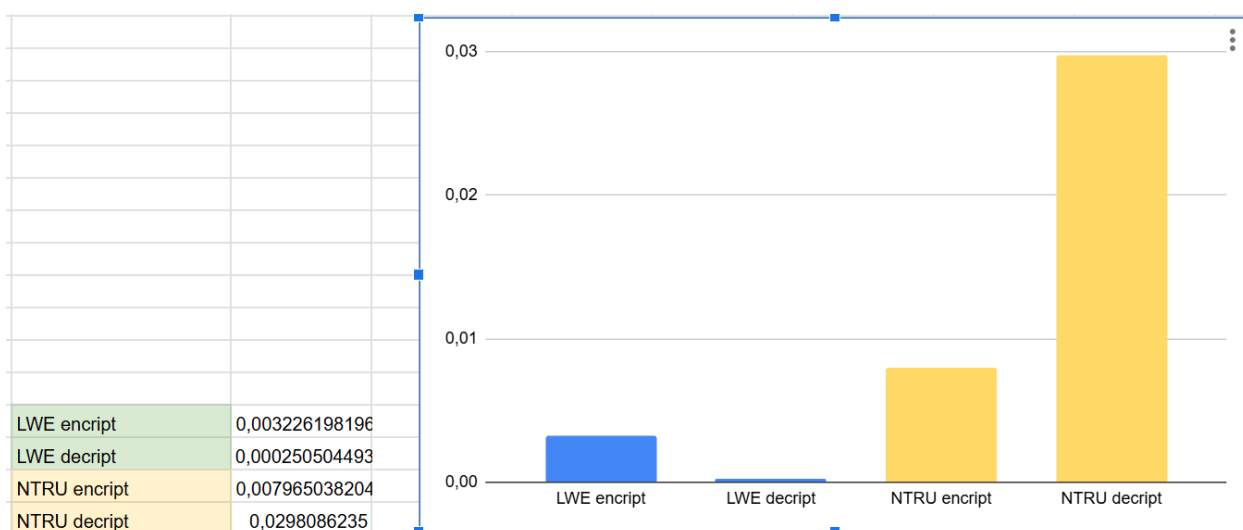


Рисунок 4.5 – Порівняння середнього часу виконання (мс)

На рисунку 4.5 представлено результати тестів для 50000 циклів. Таким чином, можна побачити, що представлена реалізація алгоритму LWE працює значно швидше. Разом з цим варто зауважити, що криптосистема NTRU показала більшу стабільність, що також є важливим фактором в оцінюванні.

4.3 Огляд варіантів

Окрім базових NTRU та LWE, у сучасних дослідженнях зустрічаються й інші оптимізовані або вдосконалені варіанти.

Ring-LWE (RLWE) – замість явної матриці A розглядається поліном $a(x)$ у кільці $Z_q[x]/(x^n+1)$ (або іншому мінімальному поліномі). Реалізація операцій множення поліномів можлива через NTT, що дає приріст у швидкодії. Ключ (секрет) – це також поліном s , шум $e(x)$. Прикладами RLWE-систем є Kyber (Key Encapsulation), гомоморфне шифрування.

Module-LWE (MLWE) – узагальнення Ring-LWE, де маємо “матриці поліномів”, забезпечує гнучкіший компроміс між розміром ключа та

швидкістю. Багато нових схем (наприклад, Dilithium – постквантовий підпис) використовують Module-LWE/SIS.

NTRU Prime – модифікація класичної NTRU, де використовується особливе просте поліномічне кільце, аби усунути деякі потенційні структури, що могли б спростити криптоаналіз. Розробники стверджують про “строгу” безпеку, оскільки кільце задано “prime” поліномом, без коренів 1 малої степені.

Усі ці варіанти спрямовані на те, щоби зменшити розміри ключів і прискорити криптографічні операції, зберігаючи безпеку, засновану на складності задач на решітках.

4.4 Практичні висновки та рекомендації

Для алгоритмів на решітках істотною є теорія коротких векторів, а також різні варіанти задачі LWE. Алгоритм LLL та його похідні (BKZ) є центральним інструментом криптоаналізу таких систем і дає змогу шукати короткі вектори в решітці. Модифікації, такі як Ring-LWE та Module-LWE, дозволяють суттєво зменшити розмір ключів і пришвидшити обчислення, зберігаючи безпеку.

NTRU – перевірена часом схема, швидка на стадії шифрування/розшифрування і з помірними ключами для рівня ~ 128 – 192 біт, але складніша для коректної (оптимізованої) реалізації ключегенерації.

LWE у чистому вигляді (з великою матрицею A) дає більші ключі; натомість його дуже зручно теоретично аналізувати і базувати на ньому різні схеми (ZK, FHE тощо). LWE у базовій формі має значно більші ключі, тоді як NTRU та Ring-LWE пропонують менший розмір.

Ring-LWE / Module-LWE – на практиці часто є золотою серединою: менші ключі, хороша швидкодія. Прикладами є Kyber (KEM), Saber (KEM), Dilithium (підписи), які NIST розглядає як стандарти.

NTRU швидко виконує операції шифрування/розшифрування, але потребує певної додаткової роботи при генерації ключів. Ring-LWE може бути дуже швидким, якщо реалізований через NTT і SIMD.

NTRU та Ring-LWE вважаються безпечними (за належних параметрів) навіть проти квантових атак, заснованих на алгоритмі Шора чи гібридних методах редукції ґраток.

У контексті стандарту NIST PQC (і загалом у промислових протоколах) уже сформувався набір “фіналістів” і “кандидатів” з числа lattice-based алгоритмів (Kyber, Dilithium, Saber, NTRU, FALCON) [13].

Подальша робота імовірно зосереджуватиметься на оптимізації обчислень (швидше множення поліномів, розподілені обчислення на GPU), а

також на захисті від сайд-каналів (пауер-аналіз, атаки через витік інформації про час виконання).

Для широкого комерційного впровадження рекомендується використовувати готові бібліотеки (наприклад, PQClean [19], Open Quantum Safe[20]) та слідувати офіційним специфікаціям, аби уникнути можливих помилок при самостійній реалізації.

Існує фундаментальна математична база, що гарантує складність задач на решітках. На практиці вже зараз доступні готові рішення для інтеграції в реальні системи.

Можливості LWE, NTRU й інших варіацій (Ring-LWE, Module-LWE) відкривають шлях до впровадження широкого спектра криптографічних сервісів (від захищеного шифрування до гомоморфних обчислень і ZK-протоколів).

Попри це, галузь продовжує активно розвиватися, тож подальші дослідження необхідні для вдосконалення алгоритмів, визначення точних параметрів безпеки та забезпечення надійного захисту проти майбутніх квантових атак.

Таким чином, криптографія на решітках є наразі одним із найперспективніших напрямів постквантових досліджень.

ВИСНОВКИ

У межах цієї дипломної роботи було досліджено криптосистеми на основі решіток, що є одним із провідних напрямів у постквантовій криптографії. Криптосистеми на основі евклідових решіток (NTRU, LWE та їхні варіації) розглядаються як одні з головних кандидатів для протидії атакам із боку квантових обчислювальних пристроїв. Задачі SVP, CVP, SIS, LWE вважаються складними навіть для квантових алгоритмів, що робить системи на решітках конкурентоспроможними порівняно з класичними RSA та ECC у постквантовому середовищі.

Для тестування криптосистем NTRU та LWE було окреслено архітектуру проєктного рішення з використанням Python та бібліотек криптосистем [15][16] LWE та NTRU. Показано швидкість шифрування та дешифрування, а також підхід до автоматизованого тестування.

За результатами тестування криптосистема LWE показала суттєво кращі швидкісні показники в обох режимах (шифрування та дешифрування). Натомість криптосистема NTRU виявилась більш стабільною в плані отримуваних результатів.

Варто зауважити, що в даному дослідженні було представлено лише одну реалізацію для кожної криптосистеми.

Порівняння криптосистем NTRU та LWE свідчить, що немає універсально “найкращої” постквантової схеми: кожна має свої переваги і недоліки залежно від галузі застосування.

Відзначимо, що під час вибору криптографічної схеми варто враховувати специфіку застосування, вимоги до безпеки та продуктивності, а також обмеження обчислювальних ресурсів. Наприклад, у випадках, де потрібна висока швидкість обробки даних, як-от у мобільних або вбудованих системах, криптосистема LWE може бути переважною завдяки своїм кращим швидкісним показникам. Проте, якщо важливою є стабільність результатів і

мінімізація варіацій у процесі шифрування та дешифрування, NTRU може бути більш підходящим вибором.

Крім того, варто враховувати можливість комбінування різних криптосистем для досягнення більшої гнучкості і безпеки. Наприклад, в окремих ситуаціях можна використовувати LWE для шифрування великих обсягів даних, а NTRU для забезпечення автентифікації і цілісності повідомлень. Такий підхід дозволяє використовувати переваги кожної з криптосистем та забезпечувати більш високий рівень безпеки.

Якщо розробник має обмежений час/ресурси, використання готової бібліотеки з реалізацією рекомендованих NIST-фіналістів заощадить багато зусиль.

У подальших дослідженнях важливо розглянути різні реалізації LWE та NTRU, а також їхні варіанти, щоб визначити найбільш оптимальні рішення для конкретних застосувань. Дослідження таких аспектів, як енергоспоживання, вимоги до пам'яті та сумісність із існуючими системами, також є критично важливими для вибору найбільш підходящої криптосистеми.

У практичних застосунках звертають увагу також на стійкість до сайд-каналів, простоту впровадження, можливість реалізації на вбудованих пристроях тощо.

Таким чином, для досягнення максимального рівня безпеки та ефективності у постквантовій криптографії необхідно враховувати різні фактори та адаптувати підхід до конкретних вимог та умов застосування. Це дозволить забезпечити надійний захист даних у світі, де квантові обчислення стають усе більш актуальними.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Regev O. On lattices, learning with errors, random linear codes, and cryptography. Baltimore MD, USA: : ACM, 2005.
2. Koshelev D. Application of Mordell-Weil lattices with large kissing numbers to acceleration of multi-scalar multiplication on elliptic curves. Lleida, 2023.
3. NTRU+Sign: compact ntru-based signatures using bimodal distribution / J. Woo et al. Seoul : Sangmyung University, 2025.
4. Albrecht M. R. Verifiable Oblivious Pseudorandom Functions from Lattices: Practical-ish and Thresholdisable. London : SandboxAQ, 2024.
5. Hsin-Yu Chiang J. Post-Quantum Threshold Ring Signature Applications from VOLE-in-the-Head. Aarhus University, 2025.
6. Lyu S. Learning With Quantization: A Ciphertext Efficient Lattice Problem with Tight Security Reduction from LWE. Jinan University, 2024.
7. Hieu Phan D. Adaptive Hardcore Bit and Quantum Key Leasing over Classical Channel from LWE with Polynomial Modulus. Paris : Institut Polytechnique de Paris, 2025.
8. Agrawal S. Time-Lock Puzzles from Lattices. Indian Institute of Technology Madras, 2025.
9. Yasuda M. A Survey of Solving SVP Algorithms and Recent Strategies for Solving the SVP Challenge. International Symposium on Mathematics, Quantum Theory, and Cryptography, 1987.
10. Hoffstein J., Pipher J., Silverman J. H. NTRU: A Ring Based Public Key Cryptosystem.
11. Gentry C. Fully homomorphic encryption using ideal lattices. Symposium on the Theory of Computing (STOC), 2009
12. Ducas L. CRYSTALS-Dilithium: A Lattice Based Digital Signature Scheme. 2018.

13. Fouque P.-A. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. 2017
14. Python. *Python.org*. URL: <https://www.python.org/> (date of access: 01.01.2025).
15. Sunil P., Abdus Samad K. GitHub - Sapphirine/ntru: NTRU Python Library with Application to Encrypted Domain. GitHub. URL: <https://github.com/Sapphirine/ntru> (date of access: 01.01.2025).
16. Teixeira M. GitHub - miguelteixeiraa/minimalist_lwe: Public Key Encryption with Learning With Errors (LWE). GitHub. URL: https://github.com/miguelteixeiraa/minimalist_lwe (date of access: 01.01.2025).
17. JavaScript Charting Library. *CanvasJS*. URL: <https://canvasjs.com/> (date of access: 01.01.2025).
18. jQuery. *jQuery*. URL: <https://jquery.com/> (date of access: 01.01.2025).
19. PQCclean. *GitHub*. URL: <https://github.com/PQCclean> (date of access: 01.01.2025).
20. Open Quantum Safe. URL: <https://openquantumsafe.org/> (date of access: 01.01.2025).

ДОДАТОК А

Текст програми

1. Зміст файлу server.js

```

const path = require('path');
const express = require('express');
const { spawn } = require('child_process');

const app = express();
const port = 3000;

function runPythonScript(scriptPath, args, callback) {

    const pythonProcess = spawn('py', [scriptPath].concat(args));

    let data = '';
    pythonProcess.stdout.on('data', (chunk) => {
        data += chunk.toString(); // Collect data from Python script
    });

    pythonProcess.stderr.on('data', (error) => {
        console.error(`stderr: ${error}`);
    });

    pythonProcess.on('close', (code) => {
        if (code !== 0) {
            console.log(`Python script exited with code ${code}`);
            callback(`Error: Script exited with code ${code}`, null);
        } else {
            //console.log('Python script executed successfully');
            callback(null, data);
        }
    });
}

app.get('/', function(req, res) {
    res.sendFile(path.join(__dirname, '/html/index.html'));
});

app.get('/pqc-test/:number', (req, res) => {
    const number = req.params.number;
    runPythonScript('compute-pqc-test.py', [number], (err, result) => {
        if (err) {
            res.status(500).send(err);
        } else {
            res.send(`${result}`);
        }
    })
}

```

```

    });
  });

  app.get('/lwe-enc/:number', (req, res) => {
    const number = req.params.number;
    runPythonScript('compute-lwe-enc.py', [number], (err, result) => {
      if (err) {
        res.status(500).send(err);
      } else {
        res.send(`${result}`);
      }
    });
  });

  app.get('/lwe-dec/:number', (req, res) => {
    const number = req.params.number;
    runPythonScript('compute-lwe-dec.py', [number], (err, result) => {
      if (err) {
        res.status(500).send(err);
      } else {
        res.send(`${result}`);
      }
    });
  });

  app.get('/ntru-enc/:number', (req, res) => {
    const number = req.params.number;
    runPythonScript('compute-ntru-enc.py', [number], (err, result) => {
      if (err) {
        res.status(500).send(err);
      } else {
        res.send(`${result}`);
      }
    });
  });

  app.get('/ntru-dec/:number', (req, res) => {
    const number = req.params.number;
    runPythonScript('compute-ntru-dec.py', [number], (err, result) => {
      if (err) {
        res.status(500).send(err);
      } else {
        res.send(`${result}`);
      }
    });
  });

  app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
  })

```

2. Зміст файлу index.html

```

<!DOCTYPE HTML>
<html>
  <head>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
    <script>
window.onload = function () {

  var updateInterval = 1000
  var dataLength = 30; // number of dataPoints visible at any point

  // CHART 1 -----
  var dps_1 = []; // dataPoints
  var chart_1 = new CanvasJS.Chart("chartContainer", {
    title :{ text: "LWE encrypt time" },
    data: [{
      type: "line",
      dataPoints: dps_1
    }]
  });

  var xVal = 0;
  var yVal = 100;

  var updateChart1 = function (yVal_local) {
    dps_1.push({
      x: xVal,
      y: yVal_local
    });
    xVal++;

    if (dps_1.length > dataLength) { dps_1.shift(); }

    chart_1.render();
  };
  // init chart 1
  updateChart1(0);

  // CHART 2 -----
  var dps_2 = []; // dataPoints
  var chart_2 = new CanvasJS.Chart("chartContainer2", {
    title :{ text: "LWE decrypt time" },
    data: [{
      type: "line",
      dataPoints: dps_2
    }]
  });

  var xVal_2 = 0;
  var yVal_2 = 100;

```

```

var updateChart2 = function (yVal_local) {
    dps_2.push({
        x: xVal_2,
        y: yVal_local
    });
    xVal_2++;

    if (dps_2.length > dataLength) { dps_2.shift(); }
    chart_2.render();
};

// init chart 2
updateChart2(0);

// CHART 3 -----
var dps_3 = []; // dataPoints
var chart_3 = new CanvasJS.Chart("chartContainer3", {
    title :{ text: "NTRU encrypt time" },
    data: [{
        type: "line",
        dataPoints: dps_3
    }]
});

var xVal_3 = 0;
var yVal_3 = 100;

var updateChart3 = function (yVal_local) {
    dps_3.push({
        x: xVal_3,
        y: yVal_local
    });
    xVal_3++;

    if (dps_3.length > dataLength) { dps_3.shift(); }
    chart_3.render();
};

// init chart 3
updateChart3(0);

// CHART 4 -----
var dps_4 = []; // dataPoints
var chart_4 = new CanvasJS.Chart("chartContainer4", {
    title :{ text: "NTRU decrypt time" },
    data: [{
        type: "line",
        dataPoints: dps_4
    }]
});

```

```

var xVal_4 = 0;
var yVal_4 = 100;

var updateChart4 = function (yVal_local) {
    dps_4.push({
        x: xVal_4,
        y: yVal_local
    });
    xVal_4++;

    if (dps_4.length > dataLength) { dps_4.shift(); }
    chart_4.render();
};

// init chart 4
updateChart4(0);

var disableButtons = function () { $('button').prop('disabled', true); }
var enableButtons = function () { $('button').prop('disabled', false); }

var submitTest = function (button_name, iterations) {
var button_obj = $('#'+ button_name);
if (button_obj.is(':disabled')) { return } else { disableButtons(); }

$.ajaxSetup({async: false});
$.ajax({
    type: "GET",
    url: '/' + button_name + '/' + iterations,
    success: function (response) {
        console.log(button_name + ': ' + response);
        // UPDATE CHART - use 1000 for milliseconds in chart
        switch (button_name) {
            case 'lwe-enc':
                updateChart1(response * 1000);
                break;
            case 'lwe-dec':
                updateChart2(response * 1000);
                break;
            case 'ntru-enc':
                updateChart3(response * 1000);
                break;
            case 'ntru-dec':
                updateChart4(response * 1000);
                break;
            default: console.log("here is no button " + button_name);
        }
        setInterval(function(){ enableButtons(); }, 10); }
});
} // window.onload

```

```

$(document).ready(function(){

    $('button').click( function(){
        // get test iterations from button ID
        var iter = this.id.split('_').pop()

        // submit test by button name
        switch (iter) {
            case '1':
                submitTest(this.name, 500);
                break;
            case '2':
                submitTest(this.name, 5000);
                break;
            case '3':
                submitTest(this.name, 50000);
                break;
            default:
                console.log ("WARNING: Iteration is not set !");
        }
    });
});
}

</script>
<style>
html,
body {
    height: 100%;
    padding: 0;
    margin: 0;
}

#div1, #div2, #div3, #div4 {
    width: 50%;
    height: 50%;
    float: left;
}

#div1 { background: #DDD; }
#div2 { background: #AAA; }
#div3 { background: #777; }
#div4 { background: #444; }
</style>
</head>
<body>
    <div id="div1">
        <div id="chartContainer" style="height: 90%;
width:100%;"></div>
        <div id="buttons_lwe_enc">
            <button type="button" name="lwe-enc"
id="test_lwe_enc_1">LWE 500</button>

```

```

        <button                type="button"                name="lwe-enc"
id="test_lwe_enc_2">LWE 5000</button>
        <button                type="button"                name="lwe-enc"
id="test_lwe_enc_3">LWE 50000</button>
        <!--button                type="button"                name="lwe-enc"
id="lwe_enc_reset">Reset</button-->
    </div>
</div>

    <div id="div2">
        <div                id="chartContainer2"                style="height:        90%;
width:100%;"></div>
        <button                type="button"                name="lwe-dec"
id="test_lwe_dec_1">LWE 500</button>
        <button                type="button"                name="lwe-dec"
id="test_lwe_dec_2">LWE 5000</button>
        <button                type="button"                name="lwe-dec"
id="test_lwe_dec_3">LWE 50000</button>
        <!--button                type="button"                name="lwe-dec"
id="lwe_dec_reset">Reset</button-->
    </div>

    <div id="div3">
        <div                id="chartContainer3"                style="height:        90%;
width:100%;"></div>
        <button                type="button"                name="ntru-enc"
id="test_ntru_enc_1">NTRU 500</button>
        <button                type="button"                name="ntru-enc"
id="test_ntru_enc_2">NTRU 5000</button>
        <button                type="button"                name="ntru-enc"
id="test_ntru_enc_3">NTRU 50000</button>
        <!--button                type="button"                name="ntru-enc"
id="ntru_enc_reset">Reset</button-->
    </div>

    <div id="div4">
        <div                id="chartContainer4"                style="height:        90%;
width:100%;"></div>
        <button                type="button"                name="ntru-dec"
id="test_ntru_dec_1">NTRU 500</button>
        <button                type="button"                name="ntru-dec"
id="test_ntru_dec_2">NTRU 5000</button>
        <button                type="button"                name="ntru-dec"
id="test_ntru_dec_3">NTRU 50000</button>
        <!--button                type="button"                name="ntru-dec"
id="ntru_dec_reset">Reset</button-->
    </div>

    <script src="https://cdn.canvasjs.com/canvasjs.min.js"></script>
</body>
</html>

```

3. Зміст файлу compute-lwe-enc.py

```
import sys
import json
from lwe.minimalist_lwe import MinimalistLWE

from random import randint
import time

if __name__ == '__main__':
    iterations = int(sys.argv[1])    # Takes number from command line
    argument

    msg = randint(1000, 2000)
    test_Time = 0

    MinimalistLWE = MinimalistLWE(97, 100)
    MinimalistLWE.gen_publicKey()
    sk = MinimalistLWE.get_privateKey()

    before_encrypt_time = time.time()
    for _ in range(iterations):
        encrypted_message = MinimalistLWE.min_encrypt(msg)
    after_encrypt_time = time.time()
    test_Time = after_encrypt_time - before_encrypt_time

    print(test_Time)
    sys.stdout.flush()
```

4. Зміст файлу compute-lwe-dec.py

```

import sys
import json
from lwe.minimalist_lwe import MinimalistLWE

from random import randint
import time

if __name__ == '__main__':
    iterations = int(sys.argv[1])    # Takes number from command line
    argument

    msg = randint(1000, 2000)
    test_Time = 0

    MinimalistLWE = MinimalistLWE(97, 100)
    MinimalistLWE.gen_publicKey()
    sk = MinimalistLWE.get_privateKey()

    encrypted_message = MinimalistLWE.min_encrypt(msg)

    before_decrypt_time = time.time()
    for _ in range(iterations):
        decrypted_message = MinimalistLWE.min_decrypt(encrypted_message,
sk)
    after_decrypt_time = time.time()
    test_Time = after_decrypt_time - before_decrypt_time

    print(test_Time)
    sys.stdout.flush()

```

5. Зміст файлу compute-ntru-enc.py

```
import sys

from random import randint
import time

if __name__ == '__main__':
    iterations = int(sys.argv[1])    # Takes number from command line
    argument

    test_Time = 0

    from ntru.ntru import *
    Bob = Ntru(7, 29, 491531)
    f = [1, 1, -1, 0, -1, 1]
    g = [-1, 0, 1, 1, 0, 0, -1]

    Bob.genPublicKey(f, g, 2)
    bob_pub_key = Bob.getPublicKey()
    Bob.setPublicKey(bob_pub_key)

    Alice = Ntru(7, 29, 491531)
    Alice.setPublicKey(bob_pub_key)
    alice_msg = [1, 0, 1, 0, 1, 1, 1]
    ranPol = [-1, -1, 1, 1]
    before_encrypt_time = time.time()
    for _ in range(iterations):
        encrypted_message = Alice.encrypt(alice_msg, ranPol)
    after_encrypt_time = time.time()
    test_Time = after_encrypt_time - before_encrypt_time

    print(test_Time)
    sys.stdout.flush()
```

6. Зміст файлу compute-ntru-dec.py

```

import sys
import json

from random import randint
import time

if __name__ == '__main__':
    iterations = int(sys.argv[1])    # Takes number from command line
    argument

    from ntru.ntru import *
    Bob = Ntru(7, 29, 491531)
    f = [1, 1, -1, 0, -1, 1]
    g = [-1, 0, 1, 1, 0, 0, -1]

    Bob.genPublicKey(f, g, 2)
    bob_pub_key = Bob.getPublicKey()
    Bob.setPublicKey(bob_pub_key)

    Alice = Ntru(7, 29, 491531)
    Alice.setPublicKey(bob_pub_key)
    alice_msg = [1, 0, 1, 0, 1, 1, 1]
    ranPol = [-1, -1, 1, 1]
    NTRU_encrypted_msg = Alice.encrypt(alice_msg, ranPol)

    before_decrypt_time = time.time()
    for _ in range(iterations):
        NTRU_decrypted_message = Bob.decrypt(NTRU_encrypted_msg)
    after_decrypt_time = time.time()
    test_Time = after_decrypt_time - before_decrypt_time

    print(test_Time)
    sys.stdout.flush()

```

ДОДАТОК Б

Слайди презентації

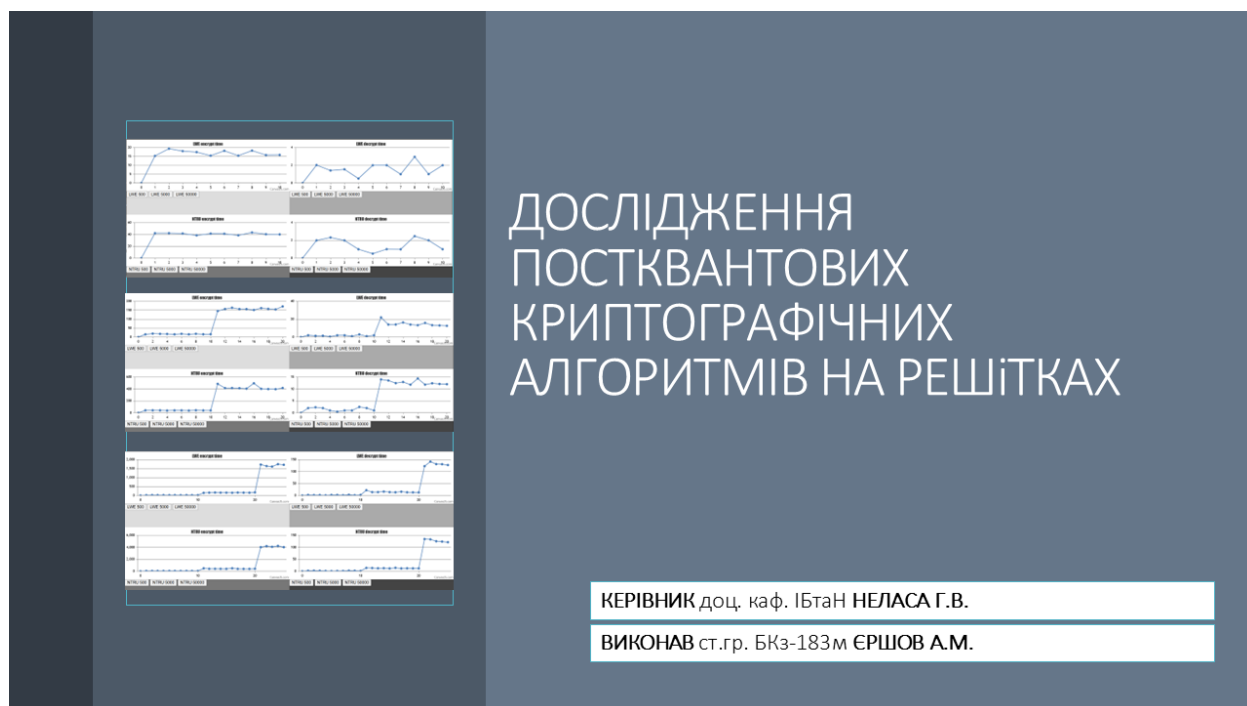


Рисунок Б.1 – Слайд 1

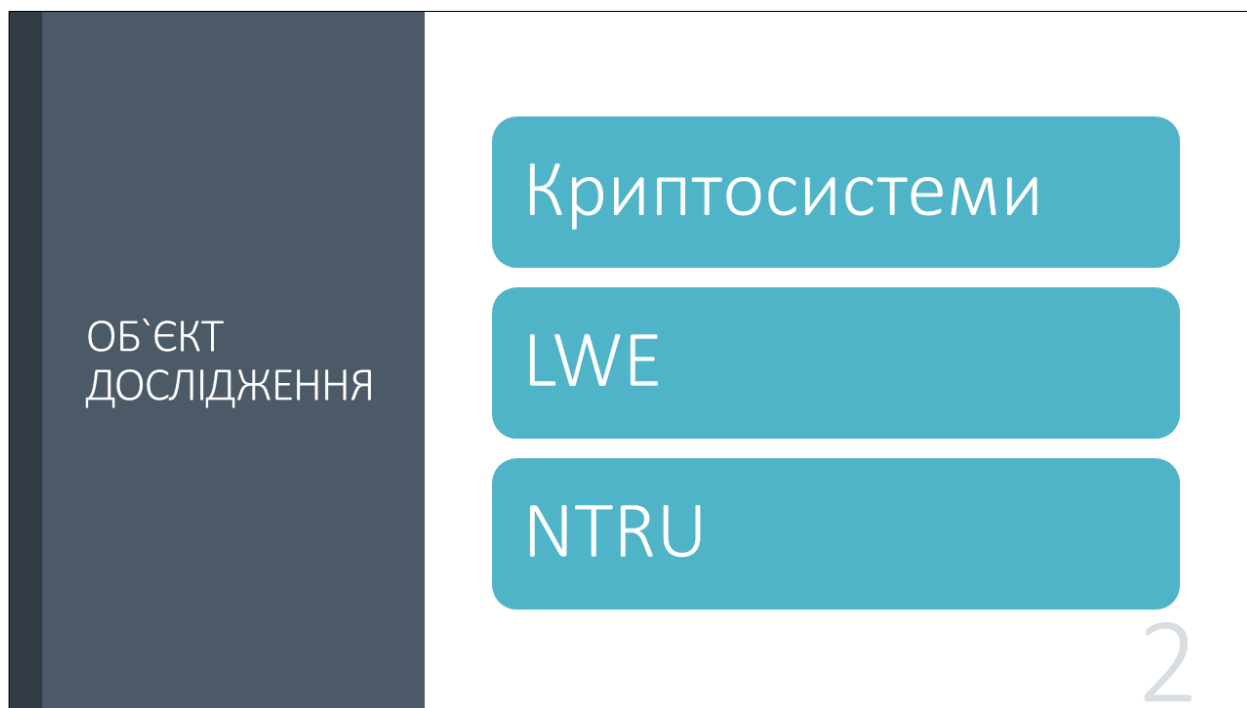


Рисунок Б.2 – Слайд 2

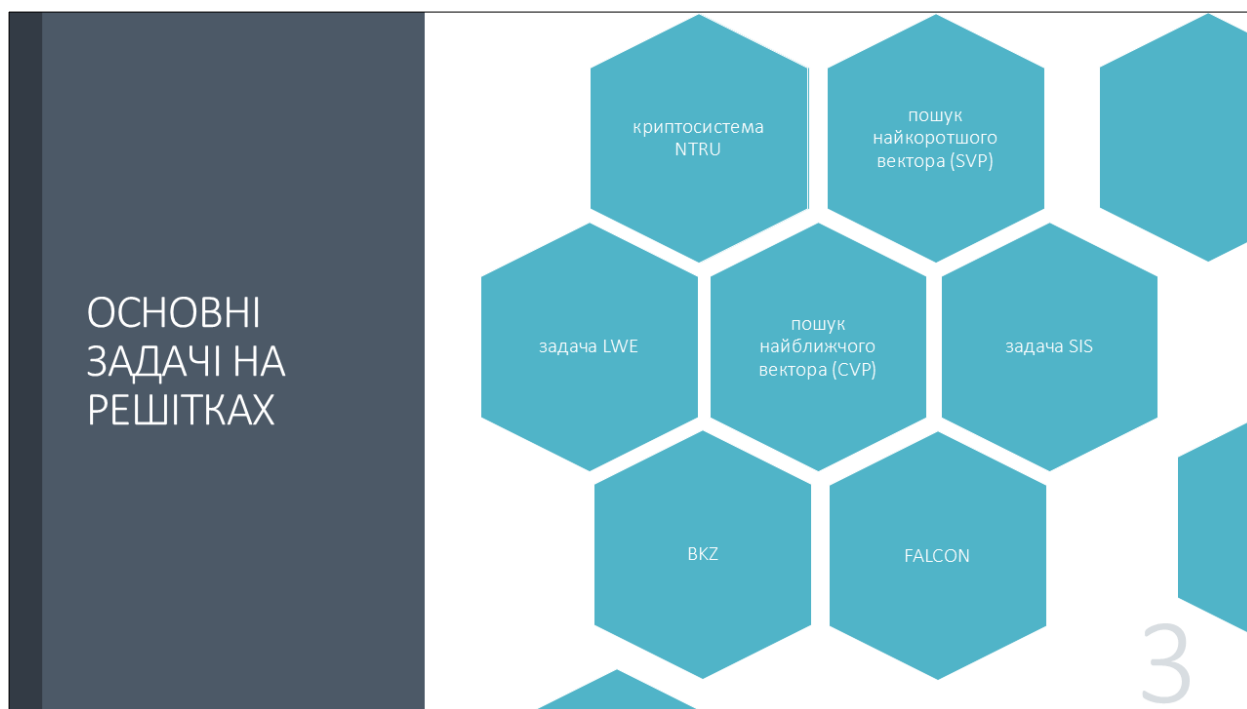


Рисунок Б.3 – Слайд 3

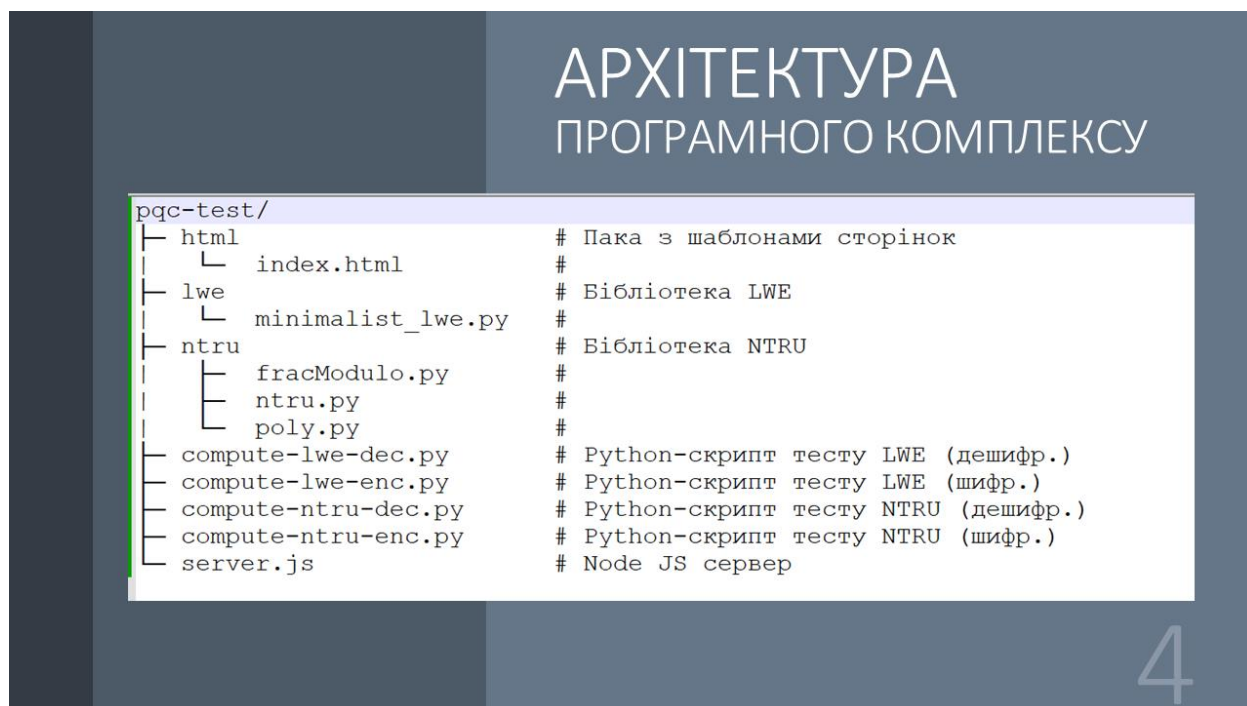


Рисунок Б.4 – Слайд 4

СХЕМАТИЧНА ДІАГРАМА ПРОГРАМНОГО КОМПЛЕКСУ

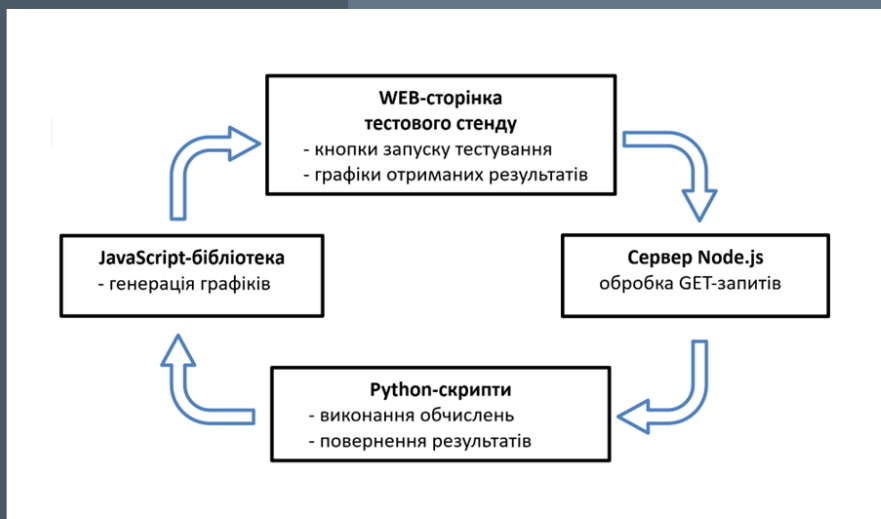


Рисунок Б.5 – Слайд 5

HTML

ОСНОВНІ КОМПОНЕНТИ

```

<!DOCTYPE HTML>
<html>
  <head>
    <script src="https://code.jquery.com/jquery-3.5.1.min.js"></script>
    <script>
      window.onload = function () {
    </script>
    <style>
  </head>
  <body>
    <div id="div1">
      <div id="chartContainer" style="height: 90%; width:100%;"></div>
      <div id="buttons_lwe_enc">
        <button type="button" name="lwe-enc" id="test_lwe_enc_1">LWE 500</button>
        <button type="button" name="lwe-enc" id="test_lwe_enc_2">LWE 5000</button>
        <button type="button" name="lwe-enc" id="test_lwe_enc_3">LWE 50000</button>
        <!--button type="button" name="lwe-enc" id="lwe_enc_reset">Reset</button-->
      </div>
    </div>
    <div id="div2">
    </div>
    <div id="div3">
    </div>
    <div id="div4">
    </div>
    <script src="https://cdn.canvasjs.com/canvasjs.min.js"></script>
  </body>
</html>
  
```

Рисунок Б.6 – Слайд 6

Node.js

ОСНОВНІ КОМПОНЕНТИ

```

const path = require('path');
const express = require('express');
const { spawn } = require('child_process');

const app = express();
const port = 3000;

function runPythonScript(scriptPath, args, callback) {
  app.get('/', function(req, res) {
    res.sendFile(path.join(__dirname, 'html/index.html'));
  });

  app.get('/pqc-test/:number', (req, res) => {
    app.get('/lwe-enc/:number', (req, res) => {
    app.get('/lwe-dec/:number', (req, res) => {
      const number = req.params.number;
      runPythonScript('compute-lwe-dec.py', [number], (err, result) => {
        if (err) {
          res.status(500).send(err);
        } else {
          res.send(`${result}`);
        }
      });
    });
  });

  app.get('/ntru-enc/:number', (req, res) => {
  app.get('/ntru-dec/:number', (req, res) => {

  app.listen(port, () => {
    console.log(`Example app listening on port ${port}`)
  })

```

7

Рисунок Б.7 – Слайд 7

Python

ОСНОВНІ КОМПОНЕНТИ

```

import sys
import json
from lwe.minimalist_lwe import MinimalistLWE

from random import randint
import time

if __name__ == '__main__':
    iterations = int(sys.argv[1]) # Takes number from command line argument

    msg = randint(1000, 2000)
    test_Time = 0

    MinimalistLWE = MinimalistLWE(97, 100)
    MinimalistLWE.gen_publicKey()
    sk = MinimalistLWE.get_privateKey()

    before_encrypt_time = time.time()
    for _ in range(iterations):
        encrypted_message = MinimalistLWE.min_encrypt(msg)
        after_encrypt_time = time.time()
        test_Time = after_encrypt_time - before_encrypt_time

    print(test_Time)
    sys.stdout.flush()

```

8

Рисунок Б.8 – Слайд 8

CanvasJS

ОСНОВНІ КОМПОНЕНТИ

```
// CHART 1 -----
var dps_1 = []; // dataPoints
var chart_1 = new CanvasJS.Chart("chartContainer", {
  title :{ text: "LWE encrypt time" },
  data: [{
    type: "line",
    dataPoints: dps_1
  }]
});

var xVal = 0;
var yVal = 100;

var updateChart1 = function (yVal_local) {
  dps_1.push({
    x: xVal,
    y: yVal_local
  });
  xVal++;

  if (dps_1.length > dataLength) { dps_1.shift(); }

  chart_1.render();
};
// init chart 1
updateChart1(0);
```

9

Рисунок Б.9 – Слайд 9

jQuery

ОСНОВНІ КОМПОНЕНТИ

```
var submitTest = function (button_name, iterations) {
  var button_obj = $('#'+ button_name);
  if (button_obj.is(':disabled')) { return } else {
    disableButtons();
  }

  $.ajaxSetup({async: false});
  $.ajax({
    type: "GET",
    url: '/' + button_name + '/' + iterations,
    success: function (response) {
      console.log(button_name + ': ' + response);
      // UPDATE CHART - use 1000 for milliseconds in chart
      switch (button_name) {
        case 'lwe-enc':
          updateChart1(response * 1000);
          break;
        case 'lwe-dec':
          updateChart2(response * 1000);
          break;
        case 'ntru-enc':
          updateChart3(response * 1000);
          break;
        case 'ntru-dec':
          updateChart4(response * 1000);
          break;
        default:

```

10

Рисунок Б.10 – Слайд 10

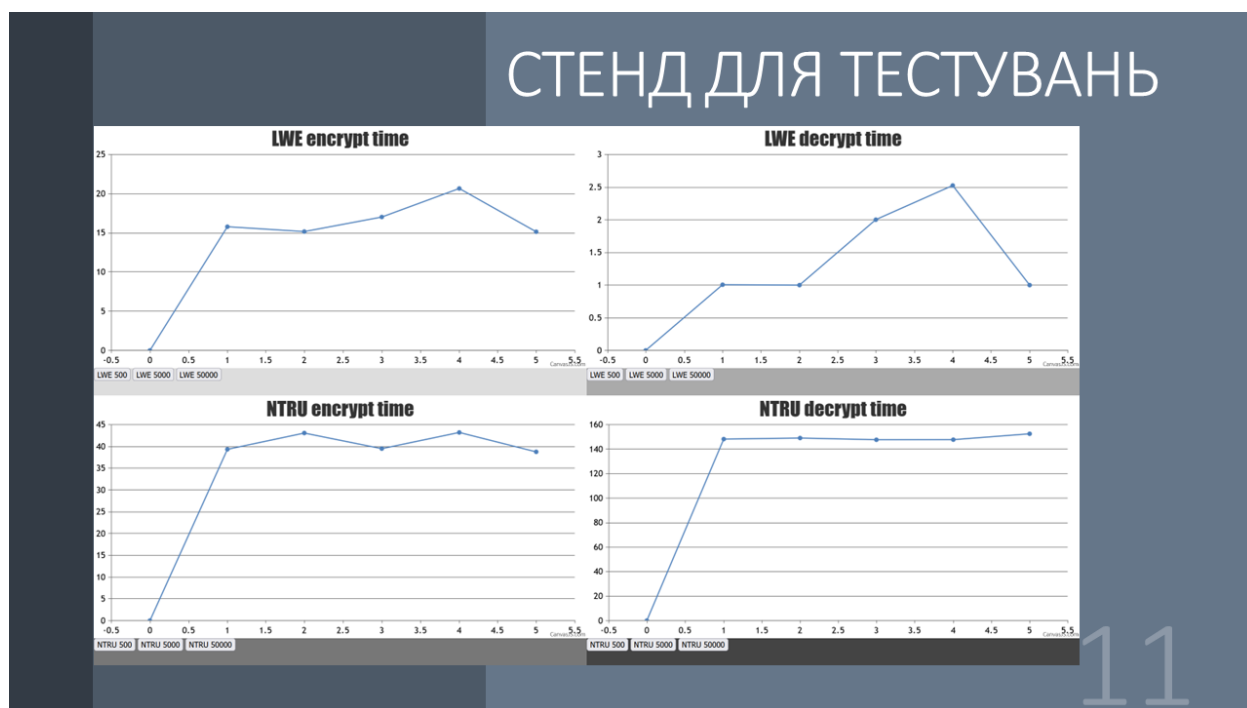


Рисунок Б.11 – Слайд 11

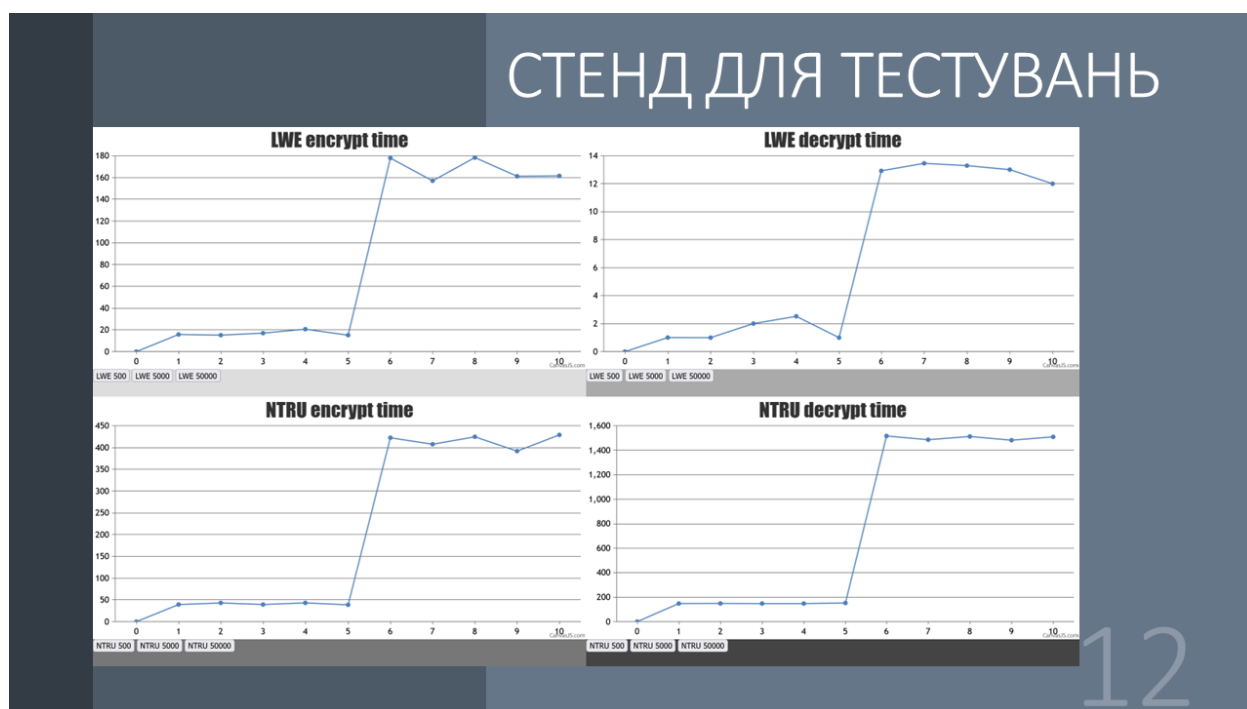


Рисунок Б.12 – Слайд 12

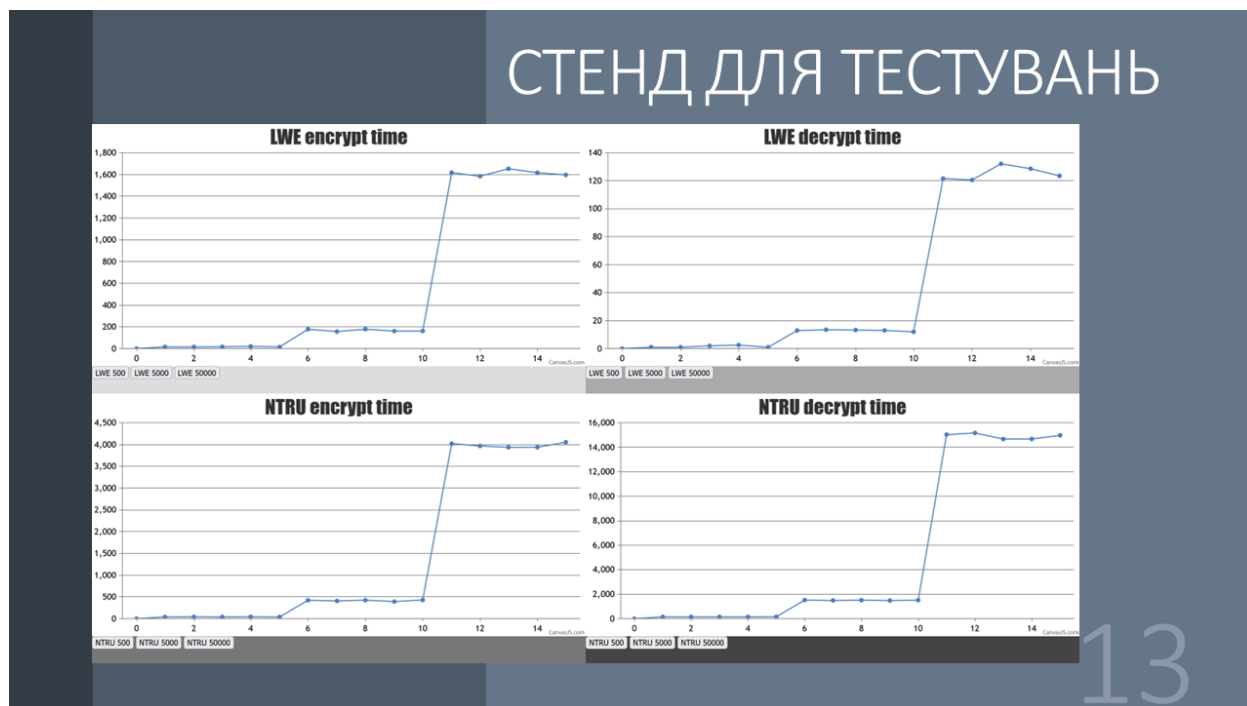


Рисунок Б.13 – Слайд 13

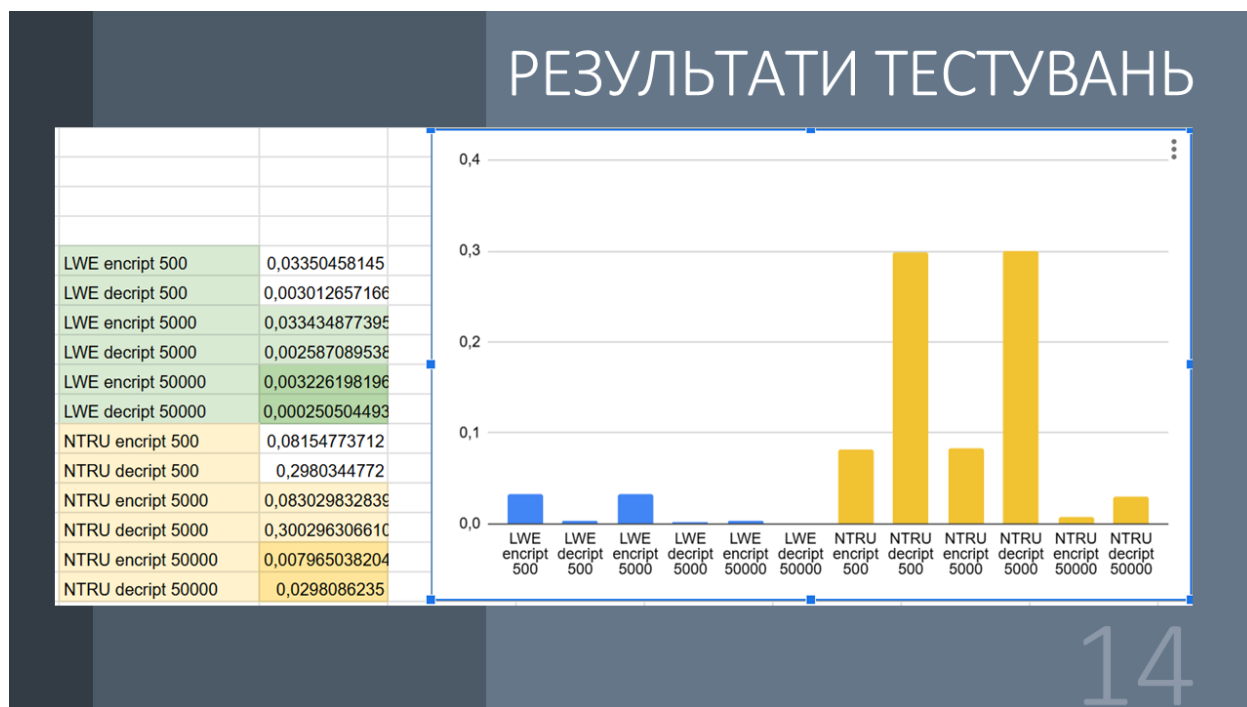


Рисунок Б.14 – Слайд 14

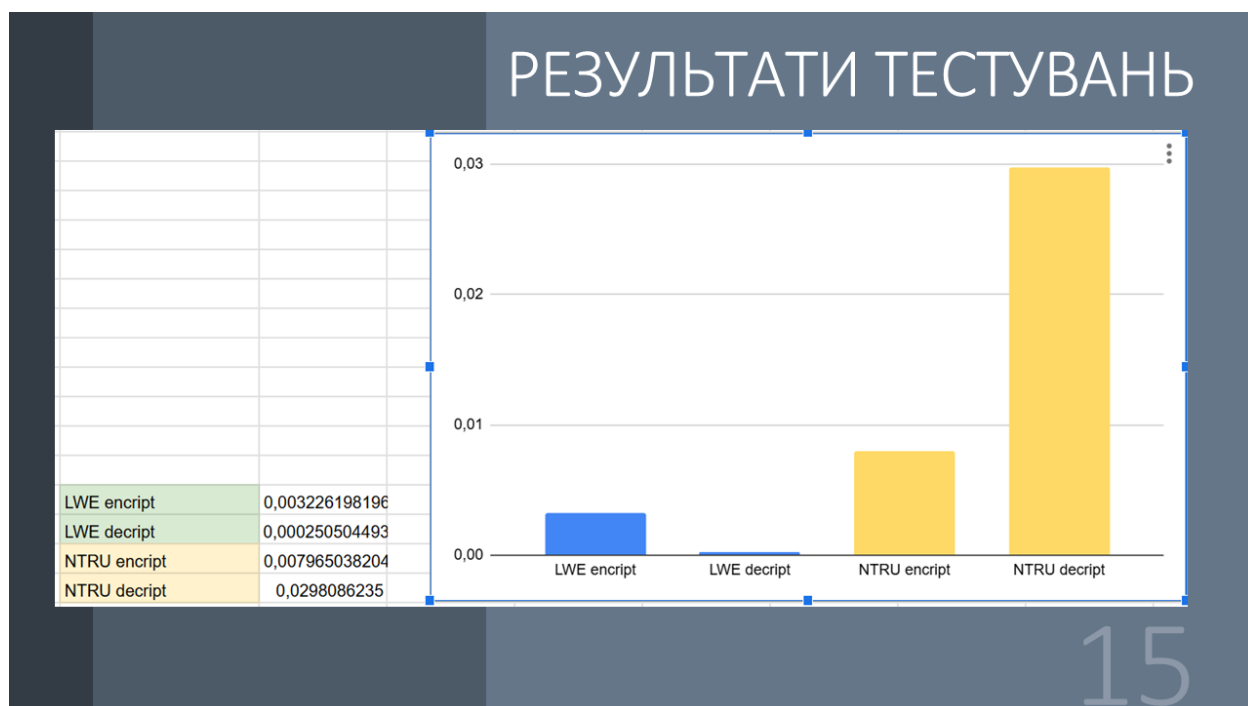


Рисунок Б.15 – Слайд 15

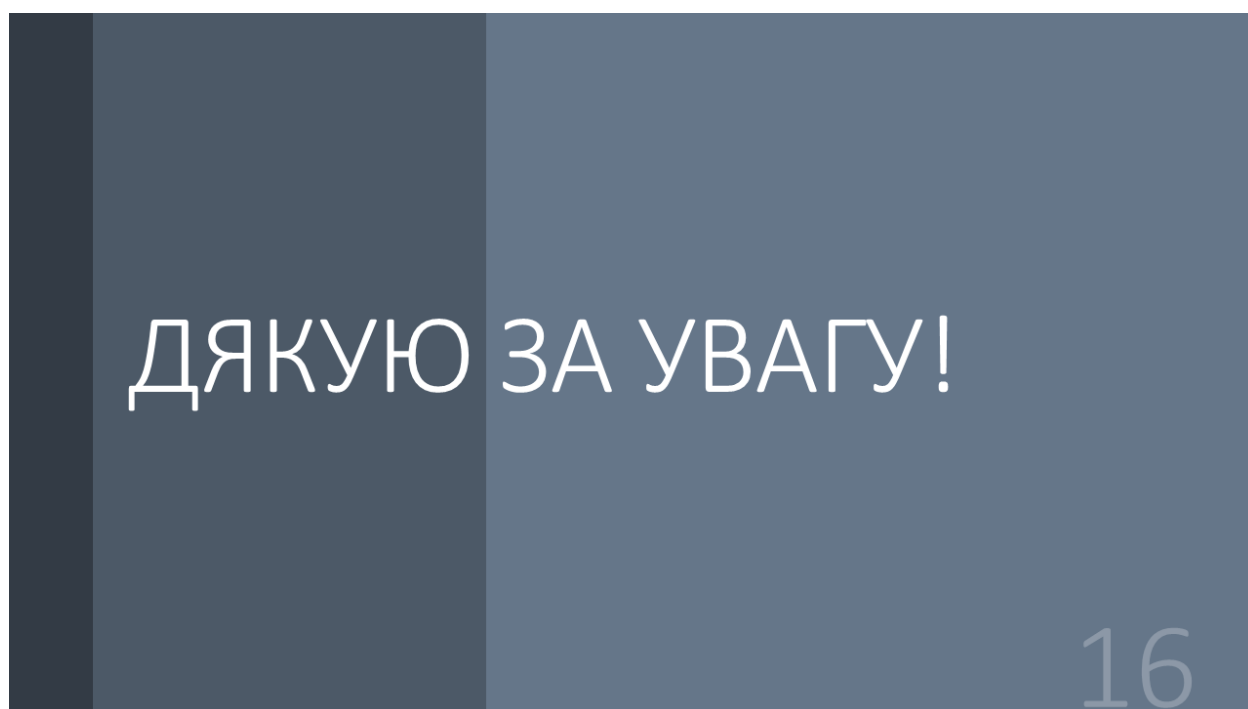


Рисунок Б.16 – Слайд 16

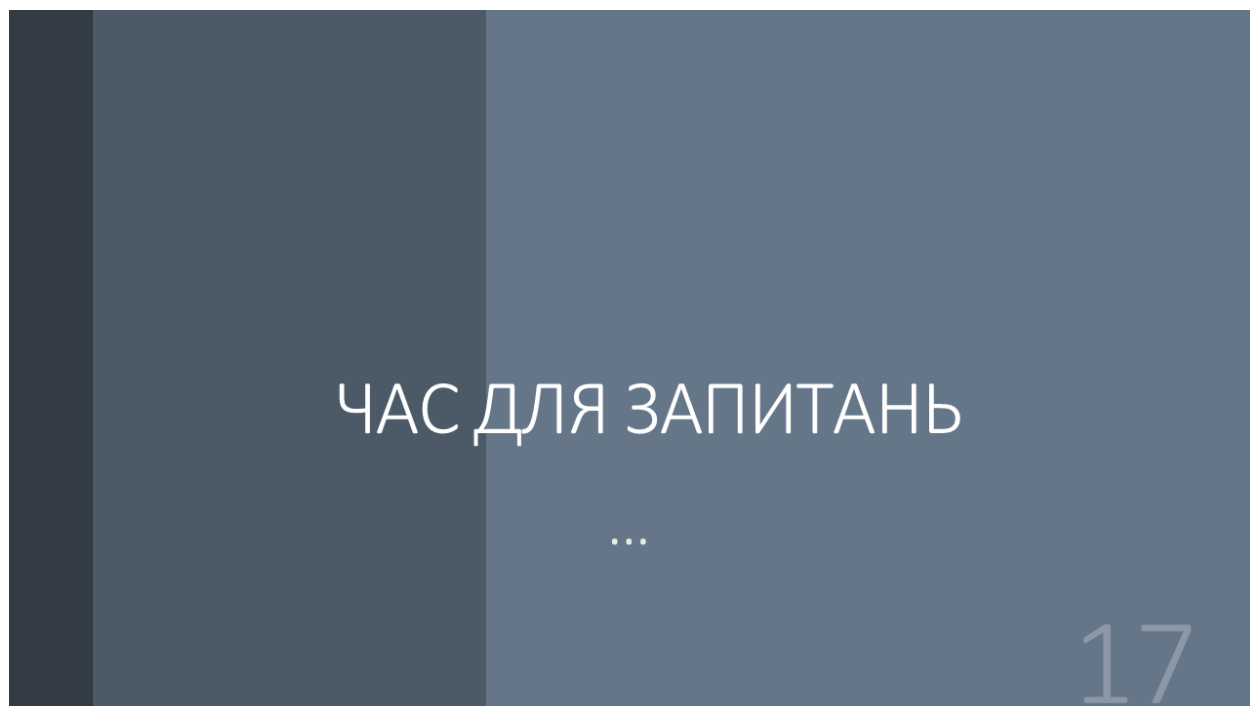


Рисунок Б.17 – Слайд 17