МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки

ЛОГОШ Вадим Дмитрович

Алгоритми гомоморфного шифрування на основі навчання з помилками / Homomorphic Encryption Algorithms Based on Learning with Error

спеціальність: 125 – Кібербезпека та захист інформації

освітньо-професійна програма – Кібербезпека

Кваліфікаційна робота

Виконав студент групи
КБм -21
В. Д. Логош

Науковий керівник
д.т.н., професор В. В.Яцків

Кваліфікаційну роботу
допущено до захисту:

«____» _____ ___2025 р.

Завідувач кафедри
_____ **В.В.Яцків**

ТЕРНОПІЛЬ - 2025

Факультет комп'ютерних інформаційних технологій
Кафедра кібербезпеки
Освітній ступінь «магістр»
спеціальність: 125 – Кібербезпека та захист інформації
освітньо-професійна програма – Кібербезпека

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____ В.В.Яцків
«____» _____ 2024 року

ЗАВДАННЯ

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

ЛОГОШ Вадим Дмитрович
(прізвище, ім'я, по батькові)

1. Тема кваліфікаційної роботи:
Алгоритми гомоморфного шифрування на основі навчання з помилками / Homomorphic Encryption Algorithms Based on Learning with Error
керівник роботи д.т.н., професор В.В. Яцків

затверджені наказом по університету від 20 грудня 2024 року № 938

2. Строк подання студентом закінченої кваліфікаційної роботи 5 грудня 2025р.

3. Вихідні дані до кваліфікаційної роботи: завдання на кваліфікаційну роботу студента, наукові статті, технічна література.

4. Основні питання, які потрібно розробити:
- проаналізувати теоретичні основи задачі навчання з помилками та її варіантів, як основи постквантових схем гомоморфного шифрування;
- виконати огляд і порівняльний аналіз сучасних алгоритмів гомоморфного шифрування на основі LWE/RLWE та відповідних бібліотек програмної реалізації;
- розробити алгоритми гомоморфного шифрування на основі LWE з підтримкою базових операцій над зашифрованими даними;
- реалізувати програмний прототип LWE-орієнтованої гомоморфної схеми та модулів для проведення експериментів;
– дослідити вплив параметрів LWE (q, n, m, σ, простір повідомлень) на ймовірність помилки розшифрування, зростання шуму та часові витрати.

5. Перелік графічного матеріалу у роботі:
– модель гомоморфного шифрування;
– основні гомоморфні схеми шифрування;
– схема шифрування на основі навчання з помилками;
– дослідження впливу параметрів на продуктивність алгоритму;
– гомоморфна операція додавання в LWE-схемі;
– результати дослідження алгоритмів гомоморфного шифрування.

6. Консультанти розділів кваліфікаційної роботи

	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання 29 листопада 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строки виконання етапів кваліфікаційної роботи	Примітка
1	Теоретичні основи гомоморфного шифрування	12.2024 р. – 03.2025 р.	
2	Розробка алгоритмів гомоморфного шифрування на основі навчання з помилками	03.2025 р. – 06.2025 р.	
3	Експериментальні дослідження ефективності алгоритмів	06.2025 р. – 11.2025 р.	

Студент _____ В. Д. Логош
(підпис)

Керівник роботи _____ В.В. Яцків
(підпис)

АНОТАЦІЯ

Логош В. А. Алгоритми гомоморфного шифрування на основі навчання з помилками. – Рукопис.

Дослідження на здобуття освітнього ступеня «магістр» за спеціальністю 125 «Кібербезпека та захист інформації», освітньо-професійна програма «Кібербезпека». – Західноукраїнський національний університет, Тернопіль, 2025.

У роботі виконано порівняльний аналіз часткових, рівневих та повних схем і сучасних бібліотек а також виявлено їхні обмеження з погляду продуктивності та глибини обчислень. На основі задачі навчання з помилками розроблено та реалізовано схему гомоморфного шифрування з підтримкою операції додавання, доповнену оптимізованими модулярними обчисленнями для зменшення часових витрат.

Ключові слова: ШИФРУВАННЯ, ГОМОМОРФНЕ ШИФРУВАННЯ, ЗАДАЧА НАВЧАННЯ З ПОМИЛКАМИ, МОДУЛЯРНІ ОБЧИСЛЕННЯ, КРИПТОАЛГОРИТМИ, ПАРАМЕТРИ БЕЗПЕКИ.

ANNOTATION

Lohosh V.A. Homomorphic Encryption Algorithms Based on Learning with Error. – Manuscript.

Thesis submitted for the degree of Master in specialty 125 “Cybersecurity and Information Protection”, educational and professional program “Cybersecurity”. – West Ukrainian National University, Ternopil, 2025.

The thesis presents a comparative analysis of partial, leveled, and fully homomorphic schemes and modern libraries, and identifies their limitations in terms of performance and computational depth.

Based on the Learning With Errors problem, a homomorphic encryption scheme with support for the addition operation is designed and implemented, complemented by optimized modular arithmetic in order to reduce computational time.

Keywords: ENCRYPTION, HOMOMORPHIC ENCRYPTION, LEARNING WITH ERRORS PROBLEM, MODULAR ARITHMETIC, CRYPTOGRAPHIC ALGORITHMS, SECURITY PARAMETERS.

ЗМІСТ

ВСТУП

Актуальність роботи. У сучасних інформаційних системах обробка конфіденційних даних усе частіше відбувається у відкритих або частково довірених середовищах – хмарних платформах, розподілених обчислювальних інфраструктурах, сервісах штучного інтелекту. У таких умовах класичні криптографічні механізми, що забезпечують лише захист даних «на зберіганні» та «під час передавання», стають недостатніми: виникає потреба в безпечній обробці даних у зашифрованому вигляді без їх розшифрування на стороні обчислювального сервісу. Саме цю задачу розв'язує гомоморфне шифрування, яке дає змогу виконувати над шифротекстами операції, що відповідають певним арифметичним або логічним операціям над відкритими даними, зберігаючи при цьому криптографічну стійкість [1, 2].

Особливе місце серед сучасних схем гомоморфного шифрування посідають алгоритми, побудовані на задачі «навчання з помилками» (Learning With Errors, LWE) та її варіантах. Ця задача належить до класу ґраткових проблем, для яких на сьогодні відсутні ефективні алгоритми розв'язання як на класичних, так і на квантових комп'ютерах, що робить LWE-підхід одним із основних кандидатів для постквантової криптографії. На основі LWE та споріднених конструкцій (RLWE, модульні ґратки тощо) побудовано низку практично орієнтованих схем часткового та повного гомоморфного шифрування, придатних для реалізації захищених хмарних сервісів, приватних обчислень, захищеного машинного навчання, електронного голосування та інших критично важливих застосувань [3].

Водночас гомоморфні схеми на основі LWE характеризуються значними обчислювальними витратами та витратами пам'яті, складною структурою параметрів та чутливістю до акумуляції шифрувального шуму. Це обумовлює актуальність наукових досліджень, спрямованих на розроблення й аналіз алгоритмів гомоморфного шифрування на основі LWE,

оптимізацію їх параметрів для конкретних прикладних сценаріїв, зменшення обчислювальної складності та покращення практичної реалізовності без втрати рівня безпеки. Додатковою мотивацією є глобальний перехід до постквантових криптографічних стандартів, ініційований, зокрема, процесом стандартизації NIST, у межах якого ґраткові та LWE-подібні схеми розглядаються як базові будівельні блоки майбутньої криптографічної інфраструктури [4].

Мета і завдання дослідження. Метою роботи є розроблення та дослідження алгоритмів гомоморфного шифрування на основі задачі навчання з помилками, орієнтованих на підвищення ефективності та забезпечення постквантової стійкості криптографічних протоколів.

Досягнення визначеної мети передбачає вирішення таких завдань:

- проаналізувати теоретичні основи задачі навчання з помилками (LWE) та її варіантів (RLWE, модульні ґратки) як основи постквантових схем гомоморфного шифрування;
- виконати огляд і порівняльний аналіз сучасних алгоритмів гомоморфного шифрування на основі LWE/RLWE та відповідних бібліотек програмної реалізації;
- сформувати алгоритмічні схеми гомоморфного шифрування на основі LWE з підтримкою базових операцій над зашифрованими даними;
- реалізувати програмний прототип LWE-орієнтованої гомоморфної схеми та модулів для проведення експериментів (шифрування, розшифрування, гомоморфні операції);
- дослідити вплив параметрів LWE (q, n, m, σ, простір повідомлень) на ймовірність помилки розшифрування, зростання шуму та часові витрати, побудувати відповідні залежності та графіки.

Об’єкт дослідження – процеси шифрування, гомоморфної обробки та розшифрування даних у криптографічних системах на основі задачі навчання з помилками.

Предмет дослідження – алгоритми гомоморфного шифрування на основі LWE/RLWE, їх параметри, методи налаштування та оцінювання.

Методи досліджень. Апарат теорії чисел, теорії ґраток, лінійної алгебри, теорії ймовірностей та математичної статистики для аналізу стійкості та поведінки шуму в LWE-схемах.

Наукова новизна одержаних результатів. Запропоновано та реалізовано методику експериментального дослідження впливу параметрів LWE-схеми (q, n, m, σ, простір повідомлень) на ймовірність помилки розшифрування, рівень шуму та часові характеристики, що дозволяє цілеспрямовано добирати параметри під задані обмеження.

Практичне значення отриманих результатів. Створено програмне забезпечення для моделювання та дослідження алгоритмів гомоморфного шифрування на основі LWE дозволяє проводити експерименти з вибору параметрів, аналізувати частоту помилок, час шифрування/дешифрування та виконання гомоморфних операцій.

Публікації та апробація КР.

1. Басістий В.П., Логош В.Д. Алгоритми гомоморфного шифрування. Матеріали X Міжнародної науково-технічної конференції «Інформаційно-комп’ютерні технології», м. Житомир, 28-29 березня 2025 р. – Житомир: Житомирська політехніка, 2025. – С.129-130.

https://conf.ztu.edu.ua/wp-content/uploads/2025/04/129.pdf

2. Логош В, Смірнов Д., Хомяк Р. Популярні бібліотеки та фреймворки гомоморфного шифрування. Матеріали проблемно-наукової міжгалузевої конференції «Кібербезпека та комп’ютерно-інтегровані технології» (КБКІТ - 2025), Тернопіль, 2025. – С. 93-95.

1 ТЕОРЕТИЧНІ ОСНОВИ ГОМОМОРФНОГО ШИФРУВАННЯ

1.1 Часткові та повні схеми гомоморфного шифрування

Гомоморфне шифрування (англ. Homomorphic Encryption, HE) – це метод шифрування, який дозволяє виконувати обчислення безпосередньо над зашифрованими даними, не розшифровуючи їх [1]. Іншими словами, якщо виконати певну операцію над шифротекстами, то після розшифрування результату ми отримаємо той самий результат, що й при виконанні цієї операції над відкритими даними. Математично це досягається шляхом побудови спеціальних криптосхем, сумісних з додаванням і множенням у зашифрованому просторі [2]. На рисунку 1.1 показано спрощену модель гомоморфного шифрування: обчислення над шифротекстами (фіолетова стрілка) еквівалентне відповідному обчисленню над відкритими даними після розшифрування [3].

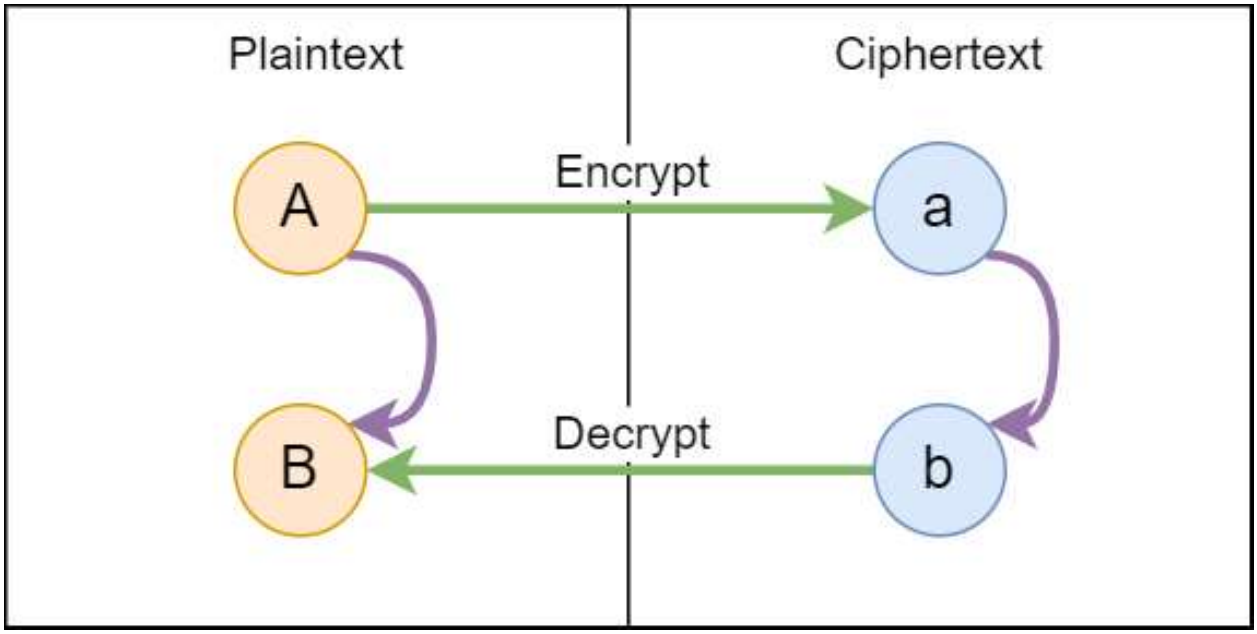

Рисунок 1.1 – Модель гомоморфного шифрування

Зашифрований вхід A перетворюється у a, виконується певне обчислення над шифротекстом (a→b), після чого розшифрування результату b дає той самий вихід B, що і пряме обчислення над відкритим текстом A.

Гомоморфне шифрування вважається “Святим Граалем” криптографії, оскільки воно теоретично дозволяє повністю захистити приватність даних під час їх використання [4]. Концепція такого шифрування була вперше озвучена ще у 1978 році Рівестом, Адельманом і Дертузосом як ідея “гомоморфізму для конфіденційності” (*privacy homomorphism*) [5]. Однак протягом тривалого часу реалізувати повністю гомоморфне шифрування (тобто з довільними обчисленнями) не вдавалося – існували лише часткові рішення.

1.1.1 Види гомоморфного шифрування

За своїми можливостями криптосхеми гомоморфного шифрування поділяються на три основні категорії [6].

Частково гомоморфне шифрування (Partial Homomorphic Encryption, PHE) – підтримує лише один тип математичної операції (наприклад, або тільки додавання, або тільки множення) у необмеженій кількості. Тобто шифросистема є гомоморфною відносно однієї операції. Такі схеми дозволяють виконувати цю операцію довільну кількість разів над шифротекстами, але не підтримують інші види операцій [7].

“Дещо” гомоморфне шифрування (Somewhat Homomorphic Encryption, SHE) – підтримує обидва типи операцій (додавання і множення), але лише у фіксованій скінченній комбінації [6]. Іншими словами, схема може виконати обмежену кількість додавань і множень над шифрованими даними (наприклад, до певної глибини арифметичного або логічного кола), після чого шифротексти стають занадто “зашумленими” для коректного розшифрування. SHE-схеми є кроком на шляху до повної гомоморфності – вони демонструють можливість обчислювати різні операції, але без необмеженого бюджету операцій [8].

Повністю гомоморфне шифрування (Fully Homomorphic Encryption, FHE) – повна гомоморфність, яка дозволяє виконувати довільні обчислення над зашифрованими даними без жодних обмежень на кількість операцій [9].

Тобто FHE-схема підтримує програмовану функціональність: можна зашифрувати вхідні дані, виконати над шифротекстами будь-яку програму (послідовність арифметичних або логічних операцій довільної довжини), і отримати зашифрований результат, який після розшифрування збігатиметься з результатом виконання цієї ж програми на відкритих даних.

У контексті практичного використання, PHE-схеми забезпечують кращу продуктивність, але дуже обмежену функціональність (тільки одна операція), тоді як FHE забезпечує універсальність, але є надзвичайно ресурсоємним на сьогодні [10, 11]. Існування проміжної категорії (SHE) історично було важливим: перш ніж з'явилося повне гомоморфне шифрування, криптографи створювали схеми, що могли виконувати хоч трохи більше, ніж одна операція, але все ще не були повністю гомоморфними [8]. Саме перетворення такої обмеженої схеми на повну гомоморфну стало ключем до прориву 2009 року.

1.1.2 Частково гомоморфні схеми шифрування

Часткове гомоморфне шифрування (PHE) підтримує одну базову операцію, тому не дозволяє виконувати довільні обчислення, але може бути корисним для спеціалізованих задач. Цікаво, що деякі класичні асиметричні криптосистеми деякі мають гомоморфні властивості [12]. Наприклад, алгоритм RSA є мультиплікативно гомоморфним: якщо зашифрувати два повідомлення m_1 і m_2 за допомогою відкритого ключа RSA, то перемноження отриманих шифротекстів еквівалентне шифруванню добутку $m_1 \cdot m_2$ тим самим ключем [13]. Це випливає з математичної формули RSA: шифрування визначається як $C = m^e \, mod \, n$ (де e – експонента відкритого ключа). Таким чином, $C(m^1) \cdot C(m^2) = \left(m^{1^e} \, mod \, n\right) \cdot \left(m^{2^e} \, mod \, n\right) \equiv (m^1 \cdot m^2)^e \, mod n$, що є шифротекстом для добутку відкритих текстів [13].

Інші відомі частково гомоморфні системи підтримують додавання. Класичним прикладом є схема Пайє (Paillier, 1999), яка адитивно гомоморфна: перемноження шифротекстів (або піднесення шифротексту до

степеня) в цій схемі призводить до додавання відповідних відкритих текстів [14, 15]. Простіше кажучи, Paillier дозволяє виконувати сумування над зашифрованими числами, але не підтримує множення. Аналогічно, система Ель-Гамаля (ElGamal) в оригінальній версії є мультиплікативно гомоморфною (підтримує множення шифротекстів), а її модифікації на еліптичних кривих – теж володіють гомоморфністю для множення. Серед інших історичних PHE-схем можна назвати криптосистеми Голдвассера–Мікалі (1982) – гомоморфна за операцією XOR (додавання по модулю 2), Бенало (1994), Окамото–Учіяма, Дамгора–Юріка та інші [16]. Всі вони забезпечують шифрування з можливістю лише однієї операції і тому не дають способу реалізувати довільні обчислення на зашифрованих даних.

Історично частково гомоморфні схеми були відомі задовго до появи FHE. Так, RSA було запропоновано ще в 1978 році, Paillier – у 1999-му, ElGamal – у 1985-му. Ці алгоритми базуються на класичних задачах теорії чисел (розклад на множники, обчислення дискретного логарифма тощо) і тому не забезпечують повної гомоморфності, оскільки підтримують тільки один тип операції [17, 18]. Проте вони стали фундаментом для перших кроків у напрямі гомоморфного шифрування. Наприклад, на основі сполучення декількох часткових схем у 2005 році була запропонована обмежено гомоморфна схема Боне–Го–Ніссіма (BGN, 2005), яка дозволяла виконати обмежену кількість операцій (декілька додавань та одне множення) завдяки використанню еліптичних кривих і парних криптосистем [18]. Це був крок до практичної реалізації ширших обчислень на шифротекстах, хоча все ще з фіксованою глибиною.

1.1.3 Повністю гомоморфне шифрування

Повністю гомоморфне шифрування (FHE) дозволяє здійснювати *довільні* обчислення над зашифрованими даними, але досягнення цього було нетривіальним. Головна проблема на шляху до FHE – це нагромадження шуму. Сучасні гомоморфні схеми шифрування (особливо ті, що підтримують

множення) вбудовують у шифротекст випадковий шум, який забезпечує їхню крипостійкість. Кожна гомоморфна операція над шифротекстами збільшує рівень цього шуму, і після певної кількості операцій шум повністю “затьмарює” корисний сигнал, роблячи неможливим відновлення відкритого тексту при розшифруванні [19]. Тому звичайна SHE-схема може виконати тільки обмежене число операцій – доки шум не перевищить допустимий поріг.

Проривом стала робота Крейга Джентрі (Craig Gentry) у 2009 році [20]. Джентрі запропонував метод, названий **“**bootstrapping**”** (від англ. “самопідйом за шнурки”), який дозволив очистити шифротекст від накопиченого шуму і тим самим виконувати необмежену кількість операцій. Ідея bootstrapping полягає у гомоморфному декодуванні: шифрування будується так, що схема може власними засобами гомоморфно виконати свій алгоритм розшифрування над зашифрованим текстом [21]. Іншими словами, маючи “трохи гомоморфну” схему (SHE), яка вже здатна обчислити невелику за глибиною функцію, Джентрі навчив її обчислювати власну функцію розшифрування, щоб скинути шум. Після такого гомоморфного розшифрування (результат якого, звісно, теж зашифрований) рівень шуму в новому шифротексті зменшується до початкового, і його можна знову використовувати для подальших обчислень [19]. Таким чином, цикл “обчислення – перезапуск (bootstrapping)” дозволив отримати повністю гомоморфну схему з необмеженою глибиною обчислень. Оригінальна схема Джентрі базувалася на математичних об’єктах, званих ідеальними ґратками, і була далеко не ефективною, проте вперше довела принципову можливість FHE [22, 21]. За словами самих криптографів, FHE перестало бути “теоретичною цікавістю” і почало перетворюватися на інструмент, придатний (хоч і з великими застереженнями) для практичного використання [23].

Подальші розробки. Після роботи Джентрі дослідники активно вдосконалювали FHE-схеми, зосередившись головним чином на нових

математичних основах – зокрема, на задачі “Learning with Errors” (LWE) та її різновидах. Практично всі сучасні FHE-схеми спираються на задачі типу “навчання з помилками”, тобто на складність розв’язання систем лінійних рівнянь з випадковим шумом [24, 25]. Особливо поширений різновид – RLWE (Ring LWE), де аналогічна складна задача сформульована в кільцях многочленів. Перевага RLWE (порівняно з базовим LWE) в тому, що він дозволяє працювати з поліноміальними шифротекстами і “пакувати” багато повідомлень в один шифротекст, виконуючи над ними паралельні однотипні операції (SIMD-підхід) [26, 27]. Важливо, що задачі на ґратках (LWE, RLWE) належать до постквантових складних задач, тобто стійкі навіть перед квантовими атаками, на відміну від RSA чи ElGamal. Таким чином, перехід до ґраткових схем не тільки відкрив шлях до FHE, а й забезпечив квантову стійкість.

Серед перших значних вдосконалень FHE були схеми, що отримали назви за ініціалами авторів: BGV-схема (Brakerski–Gentry–Vaikuntanathan) та BFV-схема (Brakerski–Fan–Vercauteren). Ці рішення (близько 2011–2012 рр.) використали ідеї Джентрі, але побудували більш ефективні шифри на основі RLWE над полями цілих чисел [28]. Вони дозволили виконувати додавання і множення над цілими числами, причому вводять поняття рівневої гомоморфності: можна заздалегідь задати максимальну глибину обчислень (рівень), на яку вистачить ресурсу шумового запасу, і виконувати всі операції без очищення шуму. Іншими словами, BGV/BFV – це leveled FHE: гомоморфність “до рівня L”. Якщо потрібно перейти за цей рівень, застосовується бустрапінг для переходу на новий цикл обчислень [29, 30]. BGV і BFV відрізняються деталями реалізації множення і механізмами управління шумом, але обидві добре підходять для точних обчислень над цілими числами [31]. На їх основі сьогодні реалізовано багато бібліотек (Microsoft SEAL, IBM HElib тощо) [26, 27].

Паралельно розвивався напрям бітових (логічних) FHE-схем, оптимізованих для роботи з булевими значеннями та двійковими векторами.

Представниками цього класу є схеми FHEW (2014 р.) та TFHE (2016 р., розроблена Chillotti та ін.) – вони працюють на рівні окремих бітів і бульових функцій [32]. Головне досягнення цих схем – надзвичайно швидке бустрапінг для кожної елементарної операції. Фактично, FHEW/TFHE виконують гомоморфне перезавантаження після кожного виходу цифрової схеми, але роблять це настільки оптимізовано, що вдається швидко обчислювати цілі бульові формули. Зокрема, TFHE здатна обробляти логічний вентиль (AND/OR/XOR) за ~0.1 мс на звичайному CPU, що хоча й повільно порівняно з наносекундами відкритих операцій, але на декілька порядків швидше ранніх схем [33]. Завдяки цьому бітові FHE добре підходять для операцій порівняння, пошуку по зашифрованих даних, реалізації умовних переходів тощо [34]. Відомо, наприклад, що на основі TFHE вдалося гомоморфно виконати навіть цілий алгоритм шифру AES або будувати таблиці пошуку (LUT) на шифрованих даних.

Ще один напрям, що виник у FHE, – це наближене гомоморфне шифрування для дійсних чисел. Класичні схеми типу BGV/BFV оперують цілими числами або скінченною арифметикою, що ускладнює реалізацію обчислень з плаваючою комою (наприклад, для машинного навчання). Проривом став алгоритм CKKS (Cheon–Kim–Kim–Song), вперше представлений на Asiacrypt 2017 [35]. Схема CKKS допускає контрольований похибковий (наближений) результат: вона кодує дійсні числа таким чином, що невелика шумова похибка трактуються як допустима неточність результату. Ідея CKKS – інтегрувати шум у plaintext, змирившись з тим, що при розшифруванні отримуємо число, близьке до справжнього результату, але з певною похибкою. Цей підхід виявився дуже ефективним для задач машинного навчання, статистики, де не потрібні абсолютно точні результати. CKKS дозволяє працювати з зашифрованими комплексними або дійсними числами, підтримує операції додавання/множення (і навіть обмежені наближення нефінітних функцій), але за рахунок контролю точності. Наразі

CKKS набув популярності як стандартний інструмент для приватних обчислень з плаваючою точкою.

Отже, сьогодні екосистема FHE складається з різних схем, оптимізованих під різні потреби: BGV/BFV – для точних цілих чис (напр., фінансові обчислення, цілочисельне MPC), TFHE/FHEW – для бульової логіки і побітових операцій (напр., шифровані порівняння, фільтрація), CKKS – для обчислень над дійсними числами з допустимою похибкою (напр., гомоморфне машинне навчання) [31]. Активно досліджуються й гібридні підходи, коли різні схеми комбінуються: наприклад, можна виконувати основні обчислення в CKKS, а окремі бітові операції (як-то порівняння) реалізувати через TFHE, зводячи результати докупи. Це розширює функціональність гомоморфних обчислень і дає розробникам гнучкість у виборі інструментів.

1.2 Алгоритмічна складність та продуктивність

Одним з головних викликів гомоморфного шифрування є його обчислювальна складність. Операції над шифротекстами на кілька порядків повільніші за ті ж операції над відкритими даними. Наприклад, перша реалізація FHE вимагала близько 30 хвилин процесорного часу для одного бітового оператора. Навіть набагато пізніші оптимізовані реалізації від IBM спочатку працювали у 10^{14} разів повільніше, ніж аналогічні відкриті обчислення, і лише після багаторічних оптимізацій вдалося досягти прискорення приблизно в 75 разів від початкового рівня (що все одно лишає величезний розрив з неопрацьованими даними). Таким чином, нинішні FHE-алгоритми залишаються ресурсомісткими: потребують великих обсягів пам’яті та часу.

З теоретичної точки зору, складність гомоморфних операцій є поліноміальною від розміру вхідних даних і параметрів. Основні операції –

гомоморфне множення і додавання – зазвичай зводяться до операцій з багато членами над великими модулями (для ґраткових схем) або до великих модульних експоненцій (для числових PHE). Проте важливою оптимізацією стало використання пакування повідомлень (batching/SIMD) – коли в один шифротекст упаковується вектор з багатьох чисел, і операції над шифротекстом здійснюються паралельно над усіма компонентами вектора. Це дозволяє амортизувати витрати: наприклад, схема BFV може в одному шифротексті обробляти сотні або тисячі 16-бітних чисел одночасно, що фактично дає прискорення на порядок у задачах лінійної алгебри, статистики тощо. Ще одне важливе досягнення – покращення асимптотичної складності: другого покоління FHE завдяки методам Бракерскі–Вайкунтанатана та ін. досягло майже оптимальної оцінки: виконання T гомоморфних операцій при безпековому параметрі k має складність $O(T \cdot polylog\ (k))$, тобто накладні витрати зростають сублінійно зі збільшенням рівня безпеки.

На практиці продуктивність гомоморфних схем визначається також необхідністю керувати шумом. Leveled-FHE підхід дозволяє налаштувати параметри під максимальну глибину обчислень і виконати всю програму без перезавантаження, що значно економить час – але глибина операцій обмежена. Натомість bootstrapping знімає обмеження на глибину, але додає великі часові накладні витрами при кожному виконанні. Сучасні схеми прагнуть зменшити цю плату: як зазначено, TFHE досягла перезавантаження менш ніж за 0.1 с на біт, що відкриває можливість виконувати його після кожної логічної операції. Тим не менш, для великих обчислень (наприклад, лінійна алгебра над матрицями, глибокі нейромережі) навіть такі показники поки недостатні – FHE залишається на порядки повільнішим, і часто практичніше обмежитись невеликою кількістю рівнів без перезавантаження.

Порівняння з нешифрованими обчисленнями. Загалом, додавання двох гомоморфно зашифрованих чисел може бути в десятки разів повільнішим за звичайне, а множення – в сотні і більше разів (залежно від схеми і параметрів). Обсяг пам’яті теж різко зростає: один шифротекст може мати

розмір від кількох сотень байт до кількох кілобайт (для порівняння, відкритий текст – кілька байтів). Крім того, ключі FHE дуже великі (публічний ключ може містити мегабайти даних у вигляді числових векторів, а секретний ключ – кілька тисяч біт). Всі ці фактори означають, що продуктивність FHE значно поступається традиційній криптографії, і на сьогодні тільки окремі задачі можуть виконуватися гомоморфно з прийнятним часом. Проте активні дослідження тривають: розробляються апаратні прискорювачі (GPU- та FPGA-рішення), спеціалізовані ASIC-чіпи для FHE, нові алгоритми швидкого множення поліномів, компресії шифротекстів тощо. Є сподівання, що у найближчі роки вдасться досягти продуктивності, достатньої для ширшого впровадження цієї технології.

1.3 Відомі обмеження і виклики

Попри вражаючий поступ, гомоморфне шифрування має низку обмежень, які стримують його широке впровадження.

1. Низька продуктивність. Швидкодія FHE значно нижча, ніж у звичайних обчисленнях. Для складних додатків (напр. обробка великих баз даних або навчання моделей ШІ) повністю гомоморфні обчислення поки що не є практичними – через повільну швидкість або проблеми з точністю FHE залишається фактично недоступним для складних комерційних застосувань. Загальний консенсус полягає в тому, що FHE наразі доцільно використовувати лише в поєднанні з іншими технологіями підвищення конфіденційності (як-от захищені багатосторонні обчислення, довірчі виконавчі середовища тощо), аби компенсувати його недоліки.

2. Розростання шуму і необхідність перезавантаження. У більшості схем після кожної операції зростає шум у шифротексті. Без спеціальних заходів це обмежує число можливих операцій. Виконання Bootstrapping хоч і вирішує проблему, але додає значні накладні витрати. Налаштування

параметрів – баланс між початковою точністю/шумом і кількістю операцій – стає нетривіальним завданням для розробників.

3. Великі розміри даних. Шифротексти в FHE займають багато місця, а операції можуть ще більше збільшувати їх розмір. Це призводить до високих вимог до пам’яті і пропускної здатності каналів зв’язку. Наприклад, зашифрований цілочисловий вектор може бути у сотні разів більшим за незашифрований. Передача і зберігання таких обсягів даних – окремий виклик.

3. Обмежений набір операцій. Хоча теоретично FHE дозволяє реалізувати довільні обчислення, на практиці не всі операції однаково ефективні. Додавання і множення – “рідні” для гомоморфних схем, а от, скажімо, порівняння двох зашифрованих чисел або обчислення нелінійної функції (напр. максимуму) вимагає побудови досить великої булевої схеми з багатьох базових операцій. Це ще більше знижує ефективність. Крім того, CKKS-схема оперує наближеними значеннями, що ускладнює контроль точності – особливо при виконанні нерівностей або циклів з невідомою кількістю ітерацій.

4. Складність реалізації. Розробка застосунків з FHE потребує спеціальних знань. Неправильний вибір параметрів може привести до втрати безпеки або до неправильних результатів (через переповнення шуму чи втрату точності). Хоча з’являються вищерівневі бібліотеки і навіть компілятори, поріг входження залишається високим: розробникам треба розуміти хоча б основи схем. Це стримує використання технології широким загалом.

Враховуючи ці обмеження, наразі FHE розглядають як перспективний, але додатковий інструмент. У багатьох сценаріях доцільно поєднувати його з іншими методами: напряму виконувати гомоморфно лише критично важливі обчислення, а менш чутливі задачі – звичайним способом. Активно досліджуються також гібридні підходи, де частина алгоритму виконується на

зашифрованих даних, а частина – із залученням користувача для розшифрування проміжних результатів.

1.4 Перспективи застосування

Попри зазначені труднощі, повністю гомоморфне шифрування відкриває якісно нові можливості для захисту даних. Зростаючі вимоги законодавства (GDPR, CCPA тощо) і ризики витоку конфіденційної інформації стимулюють інтерес до технологій, що дозволяють безпечно обробляти дані у третіх сторін. Нижче наведено кілька важливих напрямів, де FHE має перспективи.

1. Хмарні обчислення і зберігання даних. Компанії можуть передавати свої зашифровані бази даних у хмару і виконувати там обчислення, не розкриваючи дані постачальнику хмарних послуг. Це вирішує давню дилему: як використати переваги хмарних сервісів, не жертвуючи конфіденційністю. Наприклад, можна виконувати агрегацію зашифрованих фінансових транзакцій або пошук по зашифрованих записах, і лише власник даних зможе розшифрувати результат. Таке рішення усуває ризик витоку під час обробки – навіть якщо хмарний сервер скомпрометовано, дані залишаться захищеними.

2. Міжорганізаційна аналітика на конфіденційних даних. FHE дозволяє кільком сторонам спільно обчислювати деякі показники на об’єднаних даних, не розкриваючи ці дані одне одному. Наприклад, лікарні можуть сумісно аналізувати зашифровані медичні дані пацієнтів для досліджень, зберігаючи повну приватність пацієнтів. Або конкуруючі компанії можуть розрахувати зашифровано спільні статистики ринку, не видаючи своїх індивідуальних показників. Раніше для такого потрібні були довірені треті сторони або складні протоколи, тепер же можна довірити обчислення будь-якому серверу, оскільки він не бачить даних у відкритому вигляді.

3. Фінанси і блокчейн. У фінансовому секторі гомоморфне шифрування дає змогу обробляти запити до зашифрованих бухгалтерських книг, здійснювати аудити, перевіряти кредитоспроможність клієнтів тощо без доступу до сирих фінансових показників. В контексті блокчейну FHE може забезпечити конфіденційність смарт-контрактів: вузли можуть виконувати програмний код над зашифрованими транзакціями, що розв'яже проблему приватності в публічних розподілених реєстрах.

4. Приватне машинне навчання (Privacy-Preserving ML). Одним з найактивніших напрямків є застосування FHE для машинного навчання на зашифрованих даних. Зокрема, технологія дає змогу виконувати інференс нейромережі: компанія-власник моделі може приймати від користувача зашифрований вхід (наприклад, медичний знімок) і обчислити зашифрований результат (діагноз), який розшифрує лише сам користувач. При цьому ні модель не розкривається користувачу, ні його дані – власнику моделі. Вже існують прототипи таких систем для простих нейромереж (бінаризованих або невеликих багатошарових перцептронів). Крім того, гомоморфне шифрування застосовується для безпечної оцінки статистичних моделей, розрахунку ризиків, персоналізованих рекомендацій тощо – скрізь, де потрібно захистити дані клієнтів.

5. Державні та правові застосування. Влада і держоргани можуть використовувати FHE для безпечної обробки чутливої інформації. Наприклад, при підрахунку статистики по перепису населення можна агрегувати зашифровані дані громадян, не маючи можливості побачити персональні відомості. Або при перевірці дотримання податкового законодавства компанії могли б надавати зашифровані фінансові показники для автоматичної перевірки алгоритмом на відповідність критеріям – і лише у разі виявлення порушень розкривати їх для ревізора.

Необхідно зазначити, що такі сценарії наразі скоріше експериментальні – через згадані обмеження продуктивності. Однак великі технологічні компанії вкладаються в розвиток FHE, розробляючи інструменти для

спрощення його використання. Наприклад, Microsoft випустила бібліотеку SEAL, яка допомагає інтегрувати гомоморфне шифрування у прикладні рішення (вже реалізовані пілотні проекти з повністю зашифрованого аналізу даних у партнерстві з фінтех-компаніями). Google розробила інструмент Private Join and Compute для захищеного спільного аналізу даних, а також FHE Transpiler – компілятор, що перетворює звичайний код на еквівалентні гомоморфні обчислення. IBM активно працює над прискоренням HElib і пропонує хмарні сервіси з підтримкою FHE. Отже, є підстави вважати, що повністю гомоморфне шифрування поступово виходитиме за межі дослідницьких лабораторій і інтегруватиметься у реальні системи, забезпечуючи новий рівень безпеки даних.

В таблиці 1.1 наведено основні гомоморфні схеми шифрування – як часткові (PHE), так і повні (FHE). Порівнюються їх тип, підтримувані гомоморфні операції, криптографічна основа, стійкість та рік розробки [1, 2].

Таблиця 1.1 – Основні гомоморфні схеми шифрування

Схема	Тип шифрування	Підтримувані операції	Безпека (основа)	Рік
RSA	PHE (мультиплікативна)	Множення шифротекстів (без обмежень)	Факторизація цілого n (не стійка проти квантових атак)	1978
Paillier	PHE (адитивна)	Додавання шифротекстів; множення на константу	Композитний модуль (n^2); (не постквантова)	1999
Gentry FHE	FHE (повна схема)	Додавання, множення (необмежено завдяки bootstrapping)	Ідеальні ґратки (LWE) + підмножина суми; постквантова	2009
BGV	FHE (на ґратках)	Додавання, множення (рівнева або з bootstrap)	RLWE (кільцеві ґратки); постквантова	2011
BFV	FHE (на	Додавання,	RLWE (scale-invariant	2012

Схема	Тип шифрування	Підтримувані операції	Безпека (основа)	Рік
(Brakerski/Fan–Vercaut.)	ґратках)	множення (рівнева або з bootstrap)	варіант); постквантова	
TFHE (Torus FHE)	FHE (булева)	Булеві операції (бітові над шифротекстами)	Гратки GSW на торі (реал. в кільці); постквантова	2016
CKKS (Cheon – Kim et al.)	FHE (наближені обчислення)	Додавання, множення над зашифрованими дійсними числами	RLWE (дод. округлення при операціях); постквантова	2017

Пояснення до таблиці. PHE – Partial Homomorphic Encryption (часткова гомоморфність), FHE – Fully Homomorphic Encryption (повна гомоморфність). Рівневе FHE (leveled FHE) означає схему, що може працювати без bootstrapping до певної глибини операцій. Більшість сучасних FHE (BGV, BFV, CKKS) можуть працювати як рівневе (швидше, але з обмеженою глибиною) або як повні з перезавантаженням.

Пакування SIMD – можливість обробляти багатовимірні дані в одному шифротексті, значно підвищуючи продуктивність на пакетних операціях. Схема TFHE оптимізована під бінарні (логічні) схеми і тому часто використовується для побітових операцій, тоді як BGV/BFV краще підходять для цілих чисел, а CKKS – для дійсних (раціональних наближень).

Досліджено стан розвитку алгоритмів гомоморфного шифрування. Проведено порівняльний аналіз сучасних бібліотек та фреймворків гомоморфного шифрування. Розкрито можливості та обмеження, зокрема, підтримувані гомоморфні операції, криптографічна основа, та стійкість.

1.5 Аналіз бібліотек та фреймворків гомоморфного шифрування

Сьогодні існує ряд відкритих бібліотек, які реалізують схеми і надають розробникам зручні інструменти для роботи з гомоморфним шифруванням. Нижче проаналізовано найпоширеніші з них та дано коротку характеристику.

1. Microsoft SEAL. Популярна відкрита бібліотека від Microsoft, що підтримує схеми BFV та CKKS. Орієнтована на простоту використання: надає високорівневий API для виконання гомоморфних обчислень, дозволяючи будувати повністю зашифровані сховища даних і сервіси обробки без розкриття ключів. Написана на C++ (доступні обгортки для .NET, Python), оптимізована для швидкодії, має детальну документацію і приклади [26].

2. PALISADE. Бібліотека з відкритим кодом, розроблена консорціумом за підтримки DARPA. Підтримує кілька гомоморфних схем – BGV, BFV, CKKS, а також бульові TFHE/FHEW, у тому числі в багатокористувацькому (Multiparty) режимі. Відрізняється модульною архітектурою і гнучкістю налаштувань. На основі PALISADE у 2022 р. створено нову об’єднану платформу OpenFHE [27].

3. Helib. Одна з перших FHE-бібліотек, розроблена IBM (перший випуск – 2013/2014, публічний реліз – 2016 р.). Реалізує схеми BGV і CKKS (додані пізніше) та підтримує bootstrapping для BGV. Написана на C++ з відкритим кодом, HElib була націлена передусім на дослідників, надаючи гнучкий, хоч і відносно низькорівневий інтерфейс. IBM продовжує вдосконалювати HElib, зокрема оптимізуючи швидкодію [28].

4. Бібліотека TFHE. Спеціалізована бібліотека для схеми TFHE (FHE). Розроблена командою дослідників (Chillotti, Gama, Georgieva, Izabachène) і вперше опублікована у 2016–2017 рр. Орієнтована на швидкі булеві операції з частим bootstrapping. Забезпечує виконання логічних схем (AND, OR, XOR тощо) над шифрованими бітами за кілька мілісекунд. Реалізована на C++, використовує швидку операцію БПФ над тором і інші оптимізації. Підтримує

також багатокористувацький режим. Недоліком є обмеженість до побітових операцій – для роботи з великими числами чи векторами рекомендується комбінувати її з іншими бібліотеками (або використовувати гібридні підходи).

5. HEAAN (HeaAn). Відкрита бібліотека для схеми CKKS, розроблена дослідниками Сеульського національного університету (авторами CKKS). Назва розшифровується як *"*Homomorphic Encryption for Arithmetic of Approximate Numbers*"*. Перший випуск – 2016 р., згодом підтримку проекту продовжила корейська компанія CryptoLab [27]. HEAAN реалізує всі основні можливості CKKS: гомоморфні додавання, множення, масштабування, пакування/розпакування векторів. В ній однією з перших з'явилася реалізація bootstrapping для CKKS, що дозволяє виконувати необмежену кількість операцій на зашифрованих речових числах. HEAAN оптимізована для високої точності і швидкості обчислень, підтримує GPU-акселерацію для основних операцій над поліномами. Її часто використовують у дослідженнях, пов'язаних з приватними обчисленнями в AI, оскільки вона швидко впроваджує найновіші алгоритмічні вдосконалення CKKS.

6. OpenFHE. новітній фреймворк (перший реліз – липень 2022) для гомоморфного шифрування, який об'єднує напрацювання кількох попередніх бібліотек [28]. OpenFHE розроблено командою експертів з різних установ під егідою організації Duality Technologies, за участі спільноти (проект під патронатом NumFocus). Бібліотека є наступником PALISADE і включає в себе підтримку всіх основних схем: BGV, BFV, CKKS для арифметичних обчислень, а також схем TFHE і FHEW для булевих операцій. OpenFHE від початку спроектована з урахуванням можливості *bootstrapping* для всіх схем (тобто підтримує перезавантаження і для BGV/BFV/CKKS, чого раніше не було «з коробки»).

Великі технологічні компанії вкладаються в розвиток FHE, розробляючи інструменти для спрощення його використання. Наприклад, Microsoft випустила бібліотеку SEAL, яка допомагає інтегрувати гомоморфне

шифрування у прикладні рішення (вже реалізовані пілотні проекти з повністю зашифрованого аналізу даних у партнерстві з фінтех-компаніями). Google розробила інструмент Private Join and Compute для захищеного спільного аналізу даних, а також FHE Transpiler – компілятор, що перетворює звичайний код на еквівалентні гомоморфні обчислення [5]. IBM активно працює над прискоренням HElib і пропонує хмарні сервіси з підтримкою FHE. Отже, є підстави вважати, що повністю гомоморфне шифрування поступово виходитиме за межі дослідницьких лабораторій і інтегруватиметься у реальні системи, забезпечуючи новий рівень безпеки даних.

Гомоморфне шифрування вже пройшло шлях від теоретичних досліджень до реальних прототипів: існують практичні бібліотеки, перші комерційні застосунки і навіть стандарти безпеки для HE (створена у 2018 р. спільнота HomomorphicEncryption.org розробила рекомендації щодо параметрів). Часткові гомоморфні схеми (RSA, Paillier тощо) давно доступні і використовуються для окремих задач, але вони не дають універсального рішення. Повні ж FHE-схеми нині все ще обмежені в застосуванні через високу обчислювальну складність. Проте прогрес триває: дослідження в сфері FHE активно фінансуються, вдосконалюються алгоритми і апаратні прискорювачі. У найближчі роки очікується подальше скорочення розриву в продуктивності між гомоморфними та звичайними обчисленнями. Таким чином, FHE поступово наближається до того, щоб стати практичним інструментом для забезпечення конфіденційності даних у хмарних сервісах, фінансовому аналізі, медицини та інших галузях, де потрібні обчислення на захищених даних без розкриття їх змісту [36, 37].

2. РОЗРОБКА АЛГОРИТМІВ ГОМОМОРФНОГО ШИФРУВАННЯ НА ОСНОВІ НАВЧАННЯ З ПОМИЛКАМИ

2.1 Постановка задачі створення алгоритму на основі LWE

Розглянемо формалізацію задачі розробки алгоритму гомоморфного шифрування, побудованого на основі задачі навчання з помилками. Потрібно спроєктувати криптографічну схему, яка забезпечує конфіденційність даних у постквантному середовищі та підтримує виконання обмеженого набору арифметичних операцій над зашифрованими даними без розкриття відкритого тексту.

В загальному вигляді задача формулюється так. Нехай задано:

- просте або степеневе модульне кільце Z_q з параметром q;
- розмірність вектора секретного ключа n та розмірність вектора шифротексту m;
- розподіл помилки χ над Z_q, з якого вибираються “малі” шумові значення.

Потрібно:

1) визначити структуру ключів:
   - секретний ключ $s \in Z_q^n$;
   - відкритий ключ, що включає матрицю $A \in Z_q^{m \times n}$ і вектор $b = As + e \in Z_q^m$, де e – шумовий вектор з елементами, вибраними з розподілу χ;
2) описати алгоритми:
   - генерації ключів $KeyGen()$, що формує узгоджену трійку параметрів $(pk, sk, params)$;
   - шифрування $Enc_{pk}(m)$, який відображає відкритий текст m із певного простору повідомлень M у шифротекст c у просторі C, використовуючи відкритий ключ та додатковий шум;

– дешифрування $Dec_{sk}(c)$, який відновлює повідомлення m з шифротексту c за секретним ключем;

– забезпечити гомоморфні властивості схеми;

3) визначити операції $\oplus$ та $\otimes$ над шифротекстами, які відповідають додаванню та (за потреби) множенню у просторі відкритих текстів:

$$Dec_{sk}(c1 \oplus c2) = m1 + m2, Dec_{sk}(c1 \otimes c2) = m1 \cdot m2$$

для допустимого діапазону значень шуму;

4) сформулювати обмеження на глибину підтримуваної гомоморфної обчислювальної схеми (кількість послідовних операцій), щоб величина шуму не перевищувала порогове значення, при якому дешифрування залишається коректним.

Сформувати вимоги до безпеки:

1) схема має бути не менш безпечною, ніж базова задача LWE, тобто злам шифрування в обраній атакувальній моделі (наприклад, IND-CPA) повинен зводитися до розв'язання задачі LWE з відповідними параметрами (n, q, χ);

2) врахувати модель загроз: атакувальник має доступ до відкритого ключа, може спостерігати/перехоплювати шифротексти та, можливо, частково контролювати операції над ними.

Визначити вимоги до ефективності:

1) оцінити обчислювальну складність основних процедур (генерація ключів, шифрування, дешифрування, гомоморфні операції) у термінах параметрів n, m, q;

2) обмежити розміри ключів та шифротекстів таким чином, щоб схема була придатною для практичної реалізації (наприклад, для використання у хмарних сервісах чи розподілених обчисленнях).

З урахуванням наведеного, формальна постановка задачі полягає в тому, щоб на основі задачі LWE та обраних криптографічних параметрів:

- спроєктувати та описати повний набір алгоритмів KeyGen, Enc, Dec, Eval (для гомоморфних операцій);

- довести коректність (правильність відновлення повідомлень за відсутності переповнення шуму);
- обґрунтувати рівень криптостійкості відносно відомих атак на LWE;
- провести попередній аналіз обчислювальних витрат і ресурсних вимог розробленого алгоритму.

2.2 Схема шифрування на основі навчання з помилками

1. Параметри схеми

Вибираємо:

- модуль q (ціле число);
- розмір секретного вектора n (кількість його елементів);
- розмір відкритого ключа m (кількість LWE-зразків);
- розподіл шуму χ – «малі» цілі числа (наприклад, −1, 0, 1−1, 0, 1).

2. Генерація ключів (KeyGen)

1. Обираємо випадковий секретний ключ (вектор):

$$s = Z_q^n.$$

2. Обираємо випадкову матрицю:

$$A \in Z_q^{m \times n},$$

елементи – випадкові від 0 до q−1.

3. Обираємо шумовий вектор:

$$e \in Z_q^m, e_i \leftarrow \chi.$$

4. Обчислюємо:

$$b = A \cdot s + e(mod\ q),$$

(множення матриця – вектор + додавання за модулем q).

Відкритий ключ: $pk = (A, b)$.

Секретний ключ: $sk = s$.

3. Кодування одного біта повідомлення.

Повідомлення: $\mu \in \{0,1\}$.

Кодуємо його в елемент Z_q як:

$$\mu' = \begin{cases} 0, & \mu = 0, \\ \left\lfloor \frac{q}{2} \right\rfloor, & \mu = 1 \end{cases}.$$

Ідея полягає в тому що «0» кодується близько до 0, «1» – близько до $q/2$; при додаванні шуму результат залишається ближчим або до 0, або до $q/2$.

4. Шифрування (Enc).

Вхід: відкритий ключ (A, b), біт повідомлення μ.

1. Обираємо випадковий вектор:

$$x \in \{0,1\}^m$$

(часто беруть бінарний вектор для спрощення).

2. Обчислюємо:

$u = A^T x (mod\ q), u \in Z_q^n,$

$v = b^T x + \mu' + e' (mod\ q),\ e' \leftarrow \chi.$

3. Шифротекст:

$$c = (u, v).$$

5. Дешифрування

Вхід: секретний ключ s, шифротекст $c = (u, v)$.

1. Обчислюємо «прибраний» шум:

$$w = v - \langle u, s \rangle (mod\ q).$$

2. Тепер, якщо w ближче до 0 (або в «малому» інтервалі навколо 0), то $\mu = 0$;

якщо w ближче до $\left\lfloor \frac{q}{2} \right\rfloor$, то μ=1.

Формально:

$$\hat{\mu} = \begin{cases} 0, & \text{якщо } w \in \text{"околі" } 0, \\ 1, & \text{якщо } w \in \text{"околі" } \left\lfloor \frac{q}{2} \right\rfloor \end{cases}.$$

6. Перевірка

Під час шифрування:

$$b = As + e,$$

тому

$$b^T x = (A \cdot s + e)^T x = (As)^T x + e^T x.$$

При декодуванні:

$$w = v - \langle u, s \rangle = (b^T x + \mu' + e') - \langle A^T x, s \rangle = (As + e)^T x + \mu' + e' - (A^T x)^T s.$$

Але $(A\ s)^T x = (A^T x)^T s = \langle A^T x, s \rangle$,

тому:

$$w = e^T x + e' + \mu'.$$

Отже, після віднімання $\langle u, s \rangle$ лишається:

- закодоване повідомлення μ',
- плюс сумарний шум $e^T x + e'$, який малий за модулем q.

Якщо шум достатньо малий, то w усе ще буде близько до 0 (для $\mu = 0$) або близько до $\left\lfloor \frac{q}{2} \right\rfloor$ (для $\mu = 1$).

Розглянемо числовий приклад. Візьмемо для прикладу наступні параметри:

$$q = 23; n = 2; m = 3.$$

Шум χ дає значення з $\{-1, 0, 1\}$.

1. Генерація ключів.

Секретний ключ:

$$s = (3, 7) \in Z_{23}^2.$$

Відкритий ключ: генеруємо випадкову матрицю $A \in Z_{23}^{3\times 2}$,

наприклад

$$A = \begin{pmatrix} 4 & 6 \\ 5 & 2 \\ 7 & 3 \end{pmatrix}.$$

Шумовий вектор (випадково з $\{-1, 0, 1\}$:

$$e = (1, 0, -1).$$

Обчислюємо:

$$As = \begin{pmatrix} 4 & 6 \\ 5 & 2 \\ 7 & 3 \end{pmatrix} \begin{pmatrix} 3 \\ 7 \end{pmatrix} = \begin{pmatrix} 4 \cdot 3 & 6 \cdot 7 \\ 5 \cdot 3 & 2 \cdot 7 \\ 7 \cdot 3 & 3 \cdot 7 \end{pmatrix} = \begin{pmatrix} 12 + 42 \\ 15 + 14 \\ 21 + 21 \end{pmatrix} = \begin{pmatrix} 54 \\ 29 \\ 42 \end{pmatrix}.$$

Тепер беремо по модулю 23:

$54\ mod 23 = 8,\ 29\ mod 23 = 6,\ 42\ mod 23 = 19.$

Отже $As \equiv (8, 6, 19)(mod 23)$.

Додаємо шум:

$b = As + e = (8 + 1, 6 + 0, 19 + (-1)) = (9, 6, 18).$

Всі елементи вже в діапазоні $[0, 22]$, тому:

$$b = (9, 6, 18).$$

Отже:

$pk = (A, b);\ sk = s.$

2. Шифрування біта

Нехай хочемо зашифрувати $\mu = 1$.

Код:

$$\mu' = \left\lfloor \frac{q}{2} \right\rfloor = \left\lfloor \frac{23}{2} \right\rfloor = 11.$$

Обираємо випадковий бінарний вектор

$x \in \{0,1\}^3,\ x = (1, 0, 1).$

Обчислюємо:

$$u = A^T x (mod\ 23).$$

Спочатку A^T:

$$A^T = \begin{pmatrix} 4 & 5 & 7 \\ 6 & 2 & 3 \end{pmatrix}.$$

Тоді

$$u = \begin{pmatrix} 4 & 5 & 7 \\ 6 & 2 & 3 \end{pmatrix} \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \cdot 1 + 5 \cdot 0 + 7 \cdot 1 \\ 6 \cdot 1 + 2 \cdot 0 + 3 \cdot 1 \end{pmatrix} = \begin{pmatrix} 4 + 0 + 7 \\ 6 + 0 + 3 \end{pmatrix} = \begin{pmatrix} 11 \\ 9 \end{pmatrix}.$$

Оскільки 11 і 9 вже < 23, то:

$$u = (11, 9).$$

Обчислюємо

$$b^T x = (9,6,18)\begin{pmatrix}1\\0\\1\end{pmatrix} = 9\cdot 1 + 6\cdot 0 + 18\cdot 1 = 27\ mod\ 23 = 4.$$

Вибираємо додатковий шум $e' \in \{-1, 0, 1\}$, нехай $e' = 1$.

Обчислюємо:

$v = b^T x + \mu' + e' mod\ 23 = 4 + 11 + 1 = 16 mod\ 23 = 16.$

Отже шифротекст:

$$c = (u, v) = ((11{,}9),\ 16).$$

3. Дешифрування

Маємо:

$$sk = s = (3, 7),$$

$$c = (u, v) = ((11{,}9), 16).$$

Обчислюємо:

$$\langle u, s\rangle = 11\cdot 3 + 9\cdot 7 = 33 + 63 = 96.$$

Модуль 23:

$$96\ mod\ 23 = 96 - 4\cdot 23 = 96 - 92 = 4.$$

Обчислюємо:

$$w = v - \langle u, s\rangle mod\ 23 = 16 - 4 = 12\ mod\ 23 = 12.$$

Порівняємо w з 0 і з $\lfloor q/2 \rfloor = 11$.

В ідеалі, якщо б шум був 0, мали б $w = 11$.

Тут шум трохи «зсунув» значення: отримали 12, що близько до 11.

Якщо ми задаємо правило:

якщо w ближче до 0, то $\hat{\mu} = 0$;

якщо ближче до 11, то $\hat{\mu} = 1$;

так як:

відстань до 0: $|\ 12 - 0\ | = 12$;

відстань до 11: $|\ 12 - 11\ | = 1$.

Отже, декодер вирішує $\hat{\mu} = 1$, тобто правильно відновлює біт.

2.3 Гомоморфна операція додавання в LWE-схемі

Гомоморфне додавання є базовою операцією в схемах гомоморфного шифрування на основі задачі навчання з помилками. На відміну від гомоморфного множення, додавання реалізується через покомпонентне додавання шифротекстів за модулем q – і практично не створює додаткового обчислювального навантаження [10].

Розглянемо формальний опис гомоморфного додавання, його коректність та вплив на шум.

2.3.1 Визначення гомоморфного додавання

Нехай зашифровані два біти $\mu 1, \mu 2 \in \{0, 1\}$:

$$c_1 = (u_1, v_1) = Enc_{pk}(\mu_1), c_2 = (u_2, v_2) = Enc_{pk}(\mu_2).$$

Гомоморфне додавання визначається як:

$$c_{add} = c_1 \oplus c_2 = (u_{add},\ v_{add}),$$

де

$$u_{add} = u_1 + u_2 (mod\ q),$$
$$v_{add} = v_1 + v_2 (mod\ q).$$

Тобто на рівні реалізації це покомпонентне додавання шифротекстів за модулем q. Такий підхід використовується як у теоретичних побудовах FHE-схем на основі LWE, так і в практичних бібліотеках (BFV/BGV/CKKS у Microsoft SEAL та ін.) [27].

2.3.2 Алгоритм гомоморфного додавання цілих чисел

Алгоритм кодування цілого $m \in \{0, \dots, t-1\}$ складається з наступних кроків.

Крок 1. Кодування

Вибираємо t (модуль повідомлення), наприклад $t = 256$.

Визначаємо $\Delta = \lfloor q/t \rfloor$.

Крок 2. Шифрування:

$$b = \langle u, s \rangle + e + \Delta \cdot m (mod\ q)$$

(аналог v, тільки замість додавання $q/2$ для біта додаємо $\Delta \cdot m$).

Крок 3. Дешифрування:

1. Обчислюємо

$$x = b - \langle u, s \rangle (mod q).$$

2. Нормалізуємо x у інтервал $(-q/2, q/2]$.

3. Оцінюємо

$$\tilde{m} = round(x\Delta)(mod t).$$

Якщо $| etotal | < \Delta/2 |$, отримаємо правильне m.

4. Гомоморфне додавання

Тепер:

шифруємо m_1: $(ct_1) = (u_1, v_1)$;

шифрує m_2: $(ct_2) = (u_2, v_2)$;

додавання:

$(ct\_sum) = ((u_1 + u_2)\ mod\ q, (v_1 + v_2)\ mod\ q)$,

розшифровужмо як $(m_1 + m_2)\ mod\ t$.

Шум додається: $esum = e_1 + e_2$.

Додавання виконуються правильно, поки $| esum | < \Delta/2 |$.

Результат роботи алгоритму звичайного арифметичного додавання в Z_t наведений на рисунку 2.1.

```
m1 = 5
m2 = 9
ct1 = (array([ 34,  68,  74, 128, 188, 206, 128, 234]), 80)
ct2 = (array([ 72, 226,  62, 211,  56, 111,  61, 214]), 243)
ct_sum = (array([106,  37, 136,  82, 244,  60, 189, 191]), 66)
dec_sum = 14
Очікуємо (m1+m2) mod T = 14
Розшифрована сума = 14
```

Рисунок 2.1 – Результат виконання арифметичного додавання

2.3.3 Обчислювальна складність гомоморфного додавання

Нехай шифротекст LWE має вигляд $c = (u, v)$, де $u \in Z_q^n$, $v \in Z_q$.

Тоді для додавання двох шифротекстів:

- потрібно виконати n модулярних додавань для u – частини;
- одне модулярне додавання для v – частини.

Отже:

$$T \oplus= O(n)$$

модулярних додавань, що значно менше за вартість шифрування $O(n \cdot m)$ чи гомоморфного множення (часто $\tilde{O}(n^2)$ або більше) [38].

У сучасних бібліотеках FHE саме гомоморфне множення є домінуючою за часом операцією, тоді як додавання вважається «дешевим» і майже не впливає на загальний час обчислень.

Приклад. Для наочності наведемо узагальнену схему на основі спрощеного прикладу.

Нехай:

$q = 23, n = 2, \left\lfloor \frac{q}{2} \right\rfloor = 11.$

Припустимо, що зашифровані два біти $\mu_1 = 1$, $\mu_2 = 1$, і після дешифрування окремих шифротекстів маємо:

$w_1 \approx 11+$(малий шум), $w_2 \approx 11+$(малий шум).

Після гомоморфного додавання:

$$w_{add} = w_1 + w_2 \ (mod\ 23) \approx 22 + \text{(малий шум)}.$$

Тоді оцінка:

$$\widehat{m} \approx round\ \left(\frac{w_{add}}{11}\right) \approx 2,$$

що інтерпретується як $\mu_1 + \mu_2 = 2$.

Якщо ми працюємо по модулю 2 (бінарна логіка), то:

$$(\mu_1 + \mu_2) mod\ 2 = 0,$$

тобто гомоморфно реалізовано операцію XOR.

У реальних схемах замість одиничних бітів зазвичай кодують цілі числа в Z_t або вектори значень, але механізм гомоморфного додавання залишається таким самим: покомпонентне додавання шифротекстів за модулем q з контрольованим зростанням шуму [38].

2.4 Оптимізація модулярних обчислень

Як було показано в попередньому підрозділі, основні витрати в LWE- та RLWE-схемах припадають на модулярні додавання та множення над кільцем Z_q (або над кільцями поліномів з коефіцієнтами в Z_q). Для практичної реалізації гомоморфного шифрування критично важливо зменшити вартість цих операцій, оскільки вони виконуються мільйони разів у процесі шифрування, оцінки схем та перетворень. Сучасні бібліотеки (SEAL, OpenFHE, HElib та ін.) прямо покладаються на оптимізовані алгоритми модулярної арифметики та представлення чисел у СЗК для досягнення прийнятної продуктивності [26].

Розглянемо три основні напрями оптимізації:

1) застосування спеціалізованих алгоритмів модулярної редукції і множення (Барретт, Монгомері);

2) використання системи залишкових класів (СЗК);

3) зменшення кількості модулярних редукцій (так зване “ліниве” зведення за модулем).

2.4.1 Ефективні алгоритми модулярної редукції та множення

1. Класичне обчислення залишків

$$c = a(mod n)$$

або

$$c = a \cdot b(mod\ n)$$

передбачає ділення на n з отриманням частки $\lfloor a/n \rfloor$ (або $\lfloor ab/n \rfloor$), що є відносно дорогою операцією на більшості процесорів і часто не має сталого часу виконання (що також небажано з точки зору побічних каналів). Алгоритми Барретта та Монгомері замінюють ділення на n множеннями, зсувами та операціями над попередньо обчисленими константами, що суттєво пришвидшує модулярну арифметику [39].

2. Барретт-редукція. Нехай потрібно виконувати багато операцій виду

$$c = x(mod\ n),$$

де n – фіксований модуль. Алгоритм Барретта пропонує попередньо обчислити наближення до $\frac{1}{n}$ у вигляді

$$\mu = \left\lfloor \frac{b^k}{n} \right\rfloor,$$

де $b = 2$ (або інша основа системи числення), а k – достатньо велике число бітів. Тоді для довільного $x < b^k$ з добрим наближенням можна оцінити частку

$$q \approx \left\lfloor \frac{x}{n} \right\rfloor \approx \left\lfloor \frac{x \cdot \mu}{b^k} \right\rfloor,$$

і обчислити залишок

$$r = x - q \cdot n.$$

За рахунок правильно підібраних параметрів (меж для x, k) вдається гарантувати, що $r \in [0, 2n)$, тобто потрібна лише одна–дві додаткові перевірки і, за потреби, корекції $r := r - z$ [39].

Переваги Барретт-редукції:

- відсутність загального ділення на n – замінено на множення і зсуви;
- можливість повторного використання μ для великої кількості редукцій з одним і тим самим n;
- гарно підходить для випадків, коли модуль фіксований, а операцій редукції багато (типова ситуація у FHE).

У контексті LWE/RLWE-схем множення за модулем q часто реалізується як:

1) обчислення повного добутку $t = a \cdot b$ у розширеному типі (128-біт при 64-бітному a, b);
2) застосування Барретт-редукції для отримання $t\ mod\ q$.

3. Множення Монгомері

Множення Монгомері ще далі оптимізує модулярне множення за рахунок переходу до спеціального представлення чисел – “формату

Монгомері". Нехай N – модуль, а R – константа, взаємно проста з N (зазвичай $R = 2^k > N$). Тоді число a замінюють його представленням

$$\tilde{a} = aR \ mod \ N.$$

Операція REDC для деякого T визначається як

$$REDC(T) = TR^{-1} \ mod \ N,$$

де R^{-1} – обернений до R елемент за модулем N.

Алгоритм Монгомері забезпечує обчислення REDC без явного ділення на N, використовуючи лише множення, додавання та ділення на R (яке вибрано зручною степенню двійки, тому реалізується зсувом) [40].

Модульне множення в Монгомері-формі виглядає так:

1) переведення операндів у Монгомері-форму:

$$\tilde{a} = aR(mod N),$$

$$\tilde{b} = bR(mod N);$$

2) обчислення проміжного добутку:

$$T = \tilde{a} \cdot \tilde{b};$$

3) застосування REDC:

$$\tilde{c} = REDC(T) = abR(mod N);$$

4) за потреби, повернення до звичайної форми:

$$c = REDC(\tilde{c}) = ab(mod N).$$

Головний сенс у тому, що ланцюжок багатьох модулярних множень (як у піднесенні до степеня чи гомоморфному множенні) можна виконувати повністю в Монгомері-формі без повернення до звичайного представлення, а всі "дорогі" ділення на N замінено на ділення на $R = 2^k$ (зсуви).

Алгоритм має сталу структуру, що корисно і з боку захисту від таймінг-атак.

У реалізації гомоморфного шифрування підхід Монгомері застосовується для прискорення множень коефіцієнтів поліномів за модулем q, особливо у поєднанні з перетворенням NTT.

2.4.2. Застосування системи залишкових класів

Другий ключовий напрям оптимізації – представлення великих модулів у вигляді добутку кількох менших попарно взаємно простих модулів. Це і є система числення в залишках.

Нехай

$$q = q_1 q_2 \cdots q_t,$$

де q_i – попарно взаємно прості модулі (зазвичай 30– 60-бітні прості числа). Тоді елемент

$$a \in \mathbb{Z}_q$$

представляється набором залишків

$$(a_1, a_2, \dots, a_t),\ a_i = a(mod\, q_i).$$

Основна перевага: додавання та множення в $\mathbb{Z}_q$ можна виконувати покомпонентно:

$$(a_1, \dots, a_t) + (b_1, \dots, b_t) = (a_1 + b_1 mod\, q_1, \dots, a_t + b_t\ mod\, q_t),$$

$$(a_1, \dots, a_t) \cdot (b_1, \dots, b_t) = (a_1 b_1 mod\, q_1, \dots, a_t b_t\ mod\, q_t).$$

Таким чином, замість однієї “великої” модулярної операції над 200 – 800 – бітним модулем q виконується t незалежних операцій над меншими модулями q_i, які:

- значно швидші на рівні машинних інструкцій;
- можуть виконуватися паралельно (SIMD, багатопоточність);
- добре поєднуються з алгоритмами типу Барретта/Монгомері для кожного q_i. [38].

Відновлення значення a з набору залишків реалізується через китайську теорему про залишки (КТЗ), але в багатьох FHE-алгоритмах повна реконструкція потрібна рідко: більшість часу дані обробляються саме у СЗК-формі, а КТЗ використовується лише в окремих етапах (наприклад, при зміні модуля або при вивантаженні результату). У бібліотеці Microsoft SEAL, наприклад, схеми BFV і CKKS реалізовані у “Full-RNS” варіанті, де всі коефіцієнти поліномів зберігаються та обробляються в RNS-поданні.

2.5 Загальні принципи вибору параметрів для схем LWE/RLWE/MLWE

Для схем на основі LWE-типу параметри задаються трійкою (або більше):

1) розмірність:

- для класичної LWE: розмір вектора/секрету n;
- для RLWE: розмір кільця N (ступінь полінома, зазвичай $N = 2^k$);
- для MLWE: розмір модуля k та ступінь полінома N;

2) модуль q (або сукупність модулів q_i в RNS-представленні);

3) розподіл помилки: дискретна гаусіана/“псевдогаусіана” з параметром σ або невеликий дискретний розподіл;

4) додатково для FHE: допустима глибина множень (визначає сумарний модуль Q = добуток рівневих модулів).

Homomorphic Encryption Standard (HES, v1.1) дає таблиці пар (n, q), які забезпечують 128/192/256-бітну безпеку для RLWE при $\sigma \approx 3.2$. Наприклад, для уніформного секрету:

$n = 4096$:

- 128 біт: $log_2\ q \approx 111$,
- 192 біт: $log_2\ q \approx 77$,
- 256 біт: $log_2\ q \approx 60$;

n=8192:

- 128 біт: $log_2\ q \approx 220$,
- 192 біт: $log_2\ q \approx 154$,
- 256 біт: $log_2\ q \approx 120$.

Це орієнтовні значення: конкретні FHE-схеми (BFV/BGV/CKKS) зазвичай реалізують q як добуток кількох простих модулів (RNS), але сумарний $log_2\ Q$ узгоджується з цими таблицями.

Класична LWE (матриці) – приклад FrodoKEM. Консервативним еталоном (без кільцевої структури) є FrodoKEM, де LWE задається великими матрицями без структури. Специфікація пропонує три набори параметрів (таблиця 2.1).

Таблиця 2.1 – Набори параметрів для різних рівнів безпеки

Рівень безпеки (NIST)	Варіант	n	q	σ	Підтримка помилки	Прибл. рівень (AES)
Level 1	Frodo-640	640	2^{15}	2.8	[-12, … ,12]	≈ AES 128
Level 3	Frodo-976	976	2^{16}	2.3	[-10, … ,10]	≈ AES 192
Level 5	Frodo-1344	1344	2^{16}	1.4	[-6, …. ,6]	≈ AES 256

Особливості:

Безалгебраїчна LWE (звичайні матриці $n \times n$) ⇒

– великий публічний ключ (~10–20 KB і більше) та шифротексти;

– зате немає кільцевої структури, менше ризиків структурних атак.

Для таких параметрів публічна матриця $A \in \mathbb{Z}_q^{n \times m}$, секрет/помилки з дискретного розподілу, методологія вибору параметрів базується на оцінці вартості найкращих відомих атак на решітки.

Якщо потрібно LWE-модель (без RLWE) для порівнянь потрібно орієнтуватись на FrodoKEM-640/976/1344, не опускаючись нижче цих параметрів для відповідних рівнів безпеки (128/192/256 біт).

Для спроб гомоморфної обробки на базі LWE (без кілець) необхідно збільшувати q і n порівняно з Frodo, тому треба одразу використовувати RLWE/MLWE.

RLWE-параметри для FHE (BFV/BGV/CKKS). У практичних FHE-бібліотеках (SEAL, OpenFHE, HElib, Lattigo) використовують RLWE над циклотомними кільцями

$$R = \mathbb{Z}[x]/(x^N + 1), N = 2^k.$$

Орієнтири з Homomorphic Encryption Standard. HES дає для кожного N граничні значення $\boldsymbol{log_2 q}$, що забезпечують 128/192/256 біт безпеки, при $\boldsymbol{\sigma \approx 3.2.}$

Типові комбінації:

1) 128-бітний рівень:

варіант 1 (малі кільця, менше слотів, менше Q):

$N = 4096, log_2\ Q\ \approx\ 100\ -\ 120$ (кілька модулів q_i по 30 –50 біт);

варіант 2 (більше слотів, більша глибина):

$N = 8192$ чи $16384, log_2\ Q \approx 200\ -\ 300$ (для глибини множень $\sim 5 - 10$).

2) 192/256 біт: при тому ж N Standard вимагає зменшувати $log_2\ Q$, або збільшувати N, щоб підняти безпеку; для практичних FHE зазвичай переходять на:

$N = 16384$ (192 біт, середня глибина);

$N = 32768$ (прагнення до 256 біт + велика глибина).

У реальних конфігураціях CKKS/ BFV в OpenFHE/SEAL часто бачимо, наприклад: CKKS з $N = 2^{16} = 65536$, сумарний $log_2\ Q \approx 1500$ біт для глибини $\sim 10 - 12$, при цьому дотримуються 128-бітної безпеки згідно HES (оцінка відбувається на рівні "ефективного" $\boldsymbol{q}$ після врахування масштабу та глибини).

Для досліджень гомоморфних схем (BFV/BGV/CKKS) рекомендованими є наступні параметри:

1. "Легкий 128-бітний профіль" (невелика глибина, без bootstrapping):

N=4096 або 8192;

BFV/BGV: $log_2\ Q \approx 200$ (наприклад, 4 – 6 простих модулів по 30 –40 біт);

CKKS: $log_2\ Q \approx 218$ – це типовий набір для 128-бітної безпеки, який прямо рекомендують в OpenFHE для порівняння BFV/BGV/CKKS.

2. "Середній профіль 128 – 192 біт (більша глибина, без bootstrapping)":

N=16384N=16384;

$log_2 Q \approx 250\ -\ 300$ (більше рівнів множення).

3. "Високий профіль (192/256 біт, глибокі схеми або з bootstrapping)":

$N = 16384$ або 32768;

$log_2\ Q \approx 400\ -\ 800$. Тут обов'язково звірятись з оновленими рекомендаціями HES та документацією конкретної бібліотеки.

MLWE-параметри (Kyber / ML-KEM, Dilithium / ML-DSA). Для модульних решіток (MLWE) маємо вже стандартизовані параметри від NIST:

1. ML-KEM (CRYSTALS-Kyber)

FIPS 203 (ML-KEM) та специфікація Kyber задають три параметри:

ML-KEM-512 (Kyber-512)

- безпека ~128 біт (NIST level 1, ~AES-128);
- кільце: ступінь поліном $N = 256$;
- модуль $q = 3329$;
- параметр модуля $k = 2$ (модуль розмірності 2 над кільцем).

ML-KEM-768 (Kyber-768)

- безпека ~192 біт (NIST level 3, ~AES-192);
- $N = 256$, $q = 3329$, $k = 3$.

ML-KEM-1024 (Kyber-1024)

- безпека ~256 біт (NIST level 5, ~AES-256);
- $N = 256$, $q = 3329$, $k = 4$.

Фактично MLWE-екземпляр має розмірність $\sim k \cdot N$ і використовує невеликі коефіцієнти секрету/помилки, але завдяки структурі (кільце + модуль) досягає хороших розмірів ключів і швидкодії.

2. ML-DSA (CRYSTALS-Dilithium)

FIPS 204 (ML-DSA) стандартизує Dilithium як модульно-решіткову схему підпису з трьома наборами параметрів:

ML-DSA-44 (Dilithium-2) – приблизно відповідає NIST рівню 2 (між AES-128 та SHA3-256 за складністю атак);

ML-DSA-65 (Dilithium-3) – NIST рівень 3 (~AES-192);

ML-DSA-87 (Dilithium-5) – NIST рівень 5 (~AES-256).

В усіх випадках структура та масштаби параметрів сумісні з Kyber: $N = 256$, модуль q близько 8380417, модульний розмір (k, l) задають розмірність матричного відображення.

3. Використання цих параметрів як базових. Для власних MLWE-або RLWE-схем (в т.ч. гомоморфних) природно орієнтуватись на:

Kyber-768 / ML-KEM-768 як базовий 128+-бітний профіль (консервативно, за рекомендацією авторів Kyber та NIST);

ML-KEM-1024 як аналог цілі 256-бітного рівня;

– для підписів – ML-DSA-65/87 як практичні рівні 192/256 біт.

Якщо гомоморфна схема будується на MLWE, вирівнювати розмірність та модуль варто не нижче цих параметрів, а далі масштабувати q (або набір q_i) з огляду на необхідну глибину обчислень.

4. Рекомендовані набори параметрів для досліджень

Приклад можливого набіру “еталонів”, щоб при порівняльному аналізі чітко прив’язати LWE/RLWE/MLWE до сучасних рівнів безпеки:

1. LWE (plain, матрична) – консервативні орієнтири:

~128 біт: Frodo-640 ($n = 640, q = 215, \sigma = 2.8$);

~192 біт: Frodo-976 ($n = 976, q = 216, \sigma = 2.3$);

~256 біт: Frodo-1344 ($n = 1344, q = 216, \sigma = 1.4$).

2. RLWE для FHE (BFV/BGV/CKKS, без bootstrapping):

~128 біт: $N = 4096$ або 8192, $log_2 Q \approx 200$ (легкі/середні схеми);

~192 біт: N=16384, $log_2 Q \approx 250$;

~256 біт: $N = 32768$, $log_2 Q \approx 300 +$, причому параметри треба перевіряти по HES або по калькулятору конкретної бібліотеки.

3. MLWE (Kyber/ML-KEM, Dilithium/ML-DSA):

~128 біт: ML-KEM-512 (Kyber-512), ML-DSA-44;

~192 біт: ML-KEM-768, ML-DSA-65;

~256 біт: ML-KEM-1024, ML-DSA-87.

Отже, у якості базових вибраних наборів параметрів для LWE/RLWE/MLWE було обрано Frodo-640/976/1344, параметри RLWE згідно Homomorphic Encryption Standard v1.1, а також стандартизовані профілі ML-KEM-512/768/1024 та ML-DSA-44/65/87 (FIPS 203/204).

3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ ЕФЕКТИВНОСТІ АЛГОРИТМІВ

3.1 Порівняння бібліотек та фреймворків гомоморфного шифрування

Сьогодні існує ряд відкритих бібліотек, які реалізують схеми і надають розробникам зручні інструменти для роботи з гомоморфним шифруванням. Нижче проаналізовано найпоширеніші з них та дано коротку характеристику.

1. Microsoft SEAL. Популярна відкрита бібліотека від Microsoft, що підтримує схеми BFV та CKKS. Орієнтована на простоту використання: надає високорівневий API для виконання гомоморфних обчислень, дозволяючи будувати повністю зашифровані сховища даних і сервіси обробки без розкриття ключів [1]. Написана на C++ (доступні обгортки для .NET, Python), оптимізована для швидкодії, має детальну документацію і приклади.

Загальні характеристики Microsoft SEAL:

Мова: C++ (офіційні .NET-обгортки; Python-обгортки – від спільноти).

Ліцензія: відкрита, спеціальна ліцензія Microsoft Research (перейшла до більш ліберальної, типу MIT/Apache у нових версіях).

Орієнтація: простота і стабільність API, акцент на практичному використанні.

Схеми (RLWE):

BFV, BGV – гомоморфна арифметика над цілими (mod tt), класична RLWE-модель.

CKKS – приблизна арифметика над дійсними/комплексними, RLWE.

Усі схеми – СЗК-варіанти: модуль q подається як добуток менших простих модулів q_i, що дозволяє паралелізувати арифметику і не працювати з величезними 200–800-бітними модулями напряму.

Технічні особливості:

- немає зовнішніх залежностей (окрім стандартної бібліотеки C++) – легко зібрати на Windows/Linux/macOS;

- детальна інструкція з прикладами (BFV/CKKS), хороші коментарі в коді прикладів;
- основний фокус – encrypt–compute–decrypt для прикладних задач: приватна статистика, ML, обробка даних у хмарі.

Використання:

- якщо потрібнам надійна, відносно проста в освоєнні бібліотека з гарною документацією.

Для прикладних проектів, де:

- треба щось порахувати над зашифрованими цілими/дійсними;
- не потрібно глибоко лізти в тонкощі реалізації схеми.

Часто SEAL використовують як “базову реалізацію CKKS/BFV” для порівняння продуктивності інших бібліотек.

2. Helib. Одна з перших FHE-бібліотек, розроблена IBM (перший випуск – 2013/2014, публічний реліз – 2016 р.). Реалізує схеми BGV і CKKS (додані пізніше) та підтримує bootstrapping для BGV. Написана на C++ з відкритим кодом, HElib була націлена передусім на дослідників, надаючи гнучкий, хоч і відносно низькорівневий інтерфейс. IBM продовжує вдосконалювати HElib, зокрема оптимізуючи швидкодію.

Загальні характеристики HElib (IBM):

– мова: C++;

– ліцензія: Apache 2.0 (open-source від IBM);

– одна з найперших практичних реалізацій FHE (з 2013 року) – історично важлива.

Схеми (RLWE):

- BGV – з підтримкою bootstrapping;
- CKKS – approximate-number схема;

Технічні особливості.

Інтенсивно використовує бібліотеку NTL (Number Theory Library) для великої модульної арифметики.

Акцент на:

ciphertext packing (SIMD-операції над слотами);

оптимізаціях Gentry–Halevi–Smart для зменшення вартості гомоморфних операцій.

Має реалізації bootstrapping для BGV/CKKS, що історично були еталонними для досліджень з покращення продуктивності FHE.

Використання:

Для наукових експериментів з BGV/CKKS + bootstrapping, коли треба чітко розуміти, що відбувається в середині.

Як "класичну" реалізацію для порівняння з новішими бібліотеками (OpenFHE, SEAL).

Через більш низькорівневий API підходить скоріше досліднику, ніж «звичайному» розробнику веб-сервісу.

3. OpenFHE. Новітній фреймворк (перший реліз – липень 2022) для гомоморфного шифрування, який об'єднує напрацювання кількох попередніх бібліотек [4]. OpenFHE розроблено командою експертів з різних установ під егідою організації Duality Technologies, за участі спільноти (проект під патронатом NumFocus). Бібліотека є наступником PALISADE і включає в себе підтримку всіх основних схем: BGV, BFV, CKKS для арифметичних обчислень, а також схем TFHE і FHEW для булевих операцій. OpenFHE від початку спроектована з урахуванням можливості *bootstrapping* для всіх схем (тобто підтримує перезавантаження і для BGV/BFV/CKKS, чого раніше не було «з коробки») [4].

Загальні характеристики OpenFHE:

- мова: C++ (є Python-обгортки від сторонніх авторів);
- ліцензія: відкрите ПЗ (NumFOCUS, Apache-подібна модель);
- орієнтація: універсальний фреймворк для всіх основних FHE-схем.

Підтримувані схеми (усі ґрунтуються на LWE/RLWE/TLWE):

- BGV, BFV, CKKS – класичні RLWE-схеми для цілих/дійсних (в СЗК-формі).

– DM (FHEW), CGGI (TFHE) – булеві/торусні схеми для покрокового bootstrapping.

– Підтримка threshold FHE для BGV/BFV/CKKS, proxy re-encryption, мультипартійні протоколи.

Технічні особливості:

Повноцінна підтримка RLWE (кільця зі схемами BGV/BFV/CKKS, включно з:

– СЗК-представленням (розклад великого модуля на добуток простих);

– швидкою поліноміальною арифметикою через NTT.

Реалізовані різні види bootstrapping (у т.ч. approximate bootstrapping для CKKS).

Розширення для threshold / multiparty FHE: спільна генерація ключів, розподілене дешифрування тощо.

Орієнтація на відповідність Homomorphic Encryption security standard (таблиці параметрів для 128/192/256 біт).

Використання:

- якщо потрібно порівнювати різні схеми (BFV vs CKKS vs TFHE/FHEW) в одному фреймворку;
- для наукових робіт по мультипартійних протоколах, threshold FHE, проксі-перешифруванні;
- для прототипів сервісів, де потрібна не одна, а кілька моделей обчислень (цілі, дійсні, булеві).

4. Бібліотека TFHE. Спеціалізована бібліотека для схеми TFHE (FHE). Розроблена командою дослідників (Chillotti, Gama, Georgieva, Izabachène) і вперше опублікована у 2016–2017 рр. Орієнтована на швидкі булеві операції з частим bootstrapping. Забезпечує виконання логічних схем (AND, OR, XOR тощо) над шифрованими бітами за кілька мілісекунд. Реалізована на C++, використовує швидку операцію БПФ над тором і інші оптимізації. Підтримує також багатокористувацький режим. Недоліком є обмеженість до побітових

операцій – для роботи з великими числами чи векторами рекомендується комбінувати її з іншими бібліотеками (або використовувати гібридні підходи).

5. HEAAN (HeaAn). Відкрита бібліотека для схеми CKKS, розроблена дослідниками Сеульського національного університету (авторами CKKS). Назва розшифровується як *"*Homomorphic Encryption for Arithmetic of Approximate Numbers*"*. Перший випуск – 2016 р., згодом підтримку проекту продовжила корейська компанія CryptoLab [3]. HEAAN реалізує всі основні можливості CKKS: гомоморфні додавання, множення, масштабування, пакування/розпакування векторів. В ній однією з перших з'явилася реалізація bootstrapping для CKKS, що дозволяє виконувати необмежену кількість операцій на зашифрованих речових числах. HEAAN оптимізована для високої точності і швидкості обчислень, підтримує GPU-акселерацію для основних операцій над поліномами. Її часто використовують у дослідженнях, пов'язаних з приватними обчисленнями в AI, оскільки вона швидко впроваджує найновіші алгоритмічні вдосконалення CKKS [41].

Великі технологічні компанії вкладаються в розвиток FHE, розробляючи інструменти для спрощення його використання. Наприклад, Microsoft випустила бібліотеку SEAL, яка допомагає інтегрувати гомоморфне шифрування у прикладні рішення (вже реалізовані пілотні проекти з повністю зашифрованого аналізу даних у партнерстві з фінтех-компаніями). Google розробила інструмент Private Join and Compute для захищеного спільного аналізу даних, а також FHE Transpiler – компілятор, що перетворює звичайний код на еквівалентні гомоморфні обчислення [5]. IBM активно працює над прискоренням HElib і пропонує хмарні сервіси з підтримкою FHE. Отже, є підстави вважати, що повністю гомоморфне шифрування поступово виходитиме за межі дослідницьких лабораторій і інтегруватиметься у реальні системи, забезпечуючи новий рівень безпеки даних [42].

Проведено порівняльний аналіз сучасних бібліотек та фреймворків гомоморфного шифрування. Розкрито можливості та обмеження, зокрема, підтримувані гомоморфні операції, криптографічна основа, та стійкість.

3.2 Дослідження впливу параметрів на продуктивність алгоритму

Параметри n, m, q одночасно визначають і рівень безпеки (через складність задачі LWE), і продуктивність схеми. Їхній вплив можна підсумувати наступним чином.

3.2.1 Розмірність секрету n

Зі збільшенням n зростає:

- час дешифрування $T_{Dec} = O(n)$,
- час шифрування за рахунок множення матриця–вектор (лінійно по n: $O(nm)$),
- складність гомоморфного множення.

З точки зору безпеки, збільшення n робить задачу LWE суттєво складнішою (відповідно до оцінок у роботах Регева та послідовників).

Отже, n є ключовим параметром, що визначає масштабність усіх обчислень.

3.2.2 Кількість зразків m

Параметр m прямо впливає на розмір відкритого ключа ($A \in Z_q^{m \times n}$, $b \in Z_q^m$) і на час шифрування (бо $u = A^T x$ має складність $O(nm)$).

Збільшення m дає додаткову «надмірність» LWE-зразків, але практично зазвичай беруть $m = O(nlog\ q)$ або близько того, що випливає з аналізу безпеки та необхідності мати достатньо багато зразків для FHE-конструкції.

Для досліджень виберемо помірні значення m, достатні для цілей побудови ключа та гомоморфних операцій, але не надмірні, щоб не збільшувати публічний ключ.

3.2.3 Модуль q

Зростання q збільшує:

- розрядність представлення елементів,
- вартість кожної модулярної операції приблизно як $O((log\ q)^2)$.

Водночас великий q дозволяє:

- кодувати більше інформації (наприклад, кілька бітів повідомлення в одному шифротексті);
- витримувати більший шум та більшу глибину гомоморфних обчислень без bootstrapping.

На практиці використовують помірне q, але застосовують у RLWE-схемах BFV/CKKS, які дозволяють контролювати шум і зменшувати модуль у процесі обчислень.

Експеримент 1. Вплив розмірності n на час і пам'ять

При фіксованих q та відношенні $m \approx k \cdot n$ дослідити, як зростає час роботи та розміри ключів/шифротекстів при збільшенні n.

1.1. План експерименту

Постійні значення:

- модуль, наприклад $q = 2^{15} = 32768$;
- розподіл шуму (один і той самий для всіх тестів);
- відношення $m = \alpha\ n$, наприклад $\alpha \in \{1, 2\}$.

Визначаємо набір значень n:

$$n \in \{128, 256, 512, 640, 1024\}.$$

Для кожного n: генеруємо ключі (KeyGen).

Виконати:

- N_{enc} шифрувань (наприклад 10000 разів) випадкових бітів/повідомлень;

- N_{dec} дешифрувань;
- кілька гомоморфних додавань / простих обчислень.

1. Виміряти:
   - середній час $KeyGen$;
   - середній час Enc/Dec;
   - максимальний / середній час;
   - використання пам'яті (кількість байт для pk, sk, ciphertext).

1.2. Побудувати графіки:

- $T_{Enc}\ (n), T_{Dec}(n)$;
- розмір pk і ciphertext як функція n.

Порівняти з теоретичною оцінкою $O(n\,m)$ для шифрування та $O(n)$ для дешифрування (тобто показати, що час приблизно лінійно/квадратично росте з n). Зробити висновок, при яких n схема ще придатна для практичного використання (на твоєму «залізі»).

Параметри: $q\ =\ 32768, \sigma =\ 2.8, m\ =\ floor(\alpha\ *\ n), \alpha\ =\ 1.0$

$enc\_trials\ =\ 500, dec\_trials\ =\ 500.$

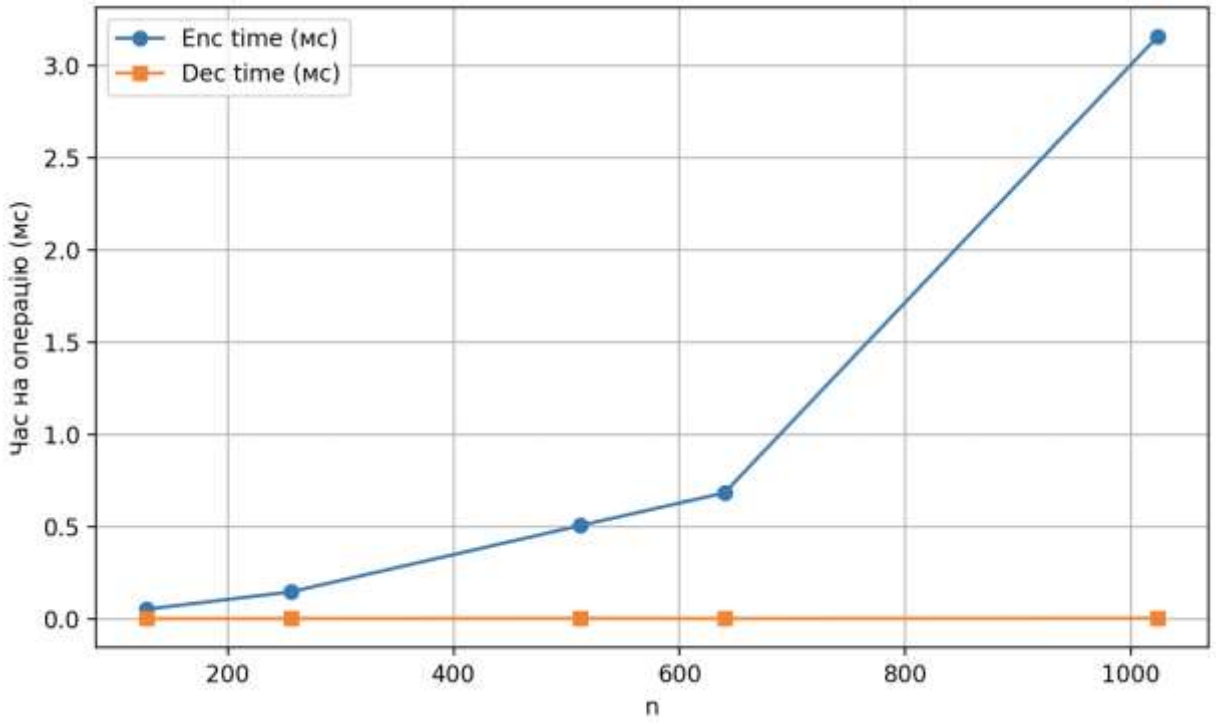

Рисунок 3.1 – Залежність часу шифрування/дешифрування від n

Таблиця 3.1 – Вплив розмірності n на час і пам’ять

Експеримент 1: Вплив розмірності n на час і пам'ять
Параметри: q = 32768, sigma = 2.8, m = floor(alpha * n), alpha = 1.0
enc_trials = 500, dec_trials = 500

n	m	T_keygen_ms	T_enc_ms	T_dec_ms	size_pk_KB	size_sk_KB	size_ct_KB
128	128	0.649	0.0492	0.0016	129.000	1.000	1.008
256	256	0.525	0.1351	0.0018	514.000	2.000	2.008
512	512	2.283	0.5110	0.0027	2052.000	4.000	4.008
640	640	3.721	0.6791	0.0020	3205.000	5.000	5.008
1024	1024	7.119	3.6216	0.0035	8200.000	8.000	8.008

З рисунку 3.1 видно, що зі збільшенням розмірності n час шифрування помітно зростає (від ~ 0.05 мс при $n = 128$ до понад 3 мс при $n \approx 1000$), тоді як час дешифрування практично не змінюється й залишається близьким до нуля на обраному масштабі. Це відповідає теоретичному аналізу: шифрування в LWE-схемі містить множення матриця–вектор розміру $m \times n$ (у нашому експерименті $m \approx n$), тобто має квадратичну залежність від n, тоді як дешифрування зводиться до одного скалярного добутку $\langle u, s \rangle$ довжини n, що дає значно менші часові витрати. Таким чином, розширення параметра n (для підвищення рівня безпеки) переважно впливає саме на вартість процедури шифрування, тоді як дешифрування залишається обчислювально дешевою операцією.

Рисунок 3.2 демонструє різке зростання розміру відкритого ключа LWE-схеми зі збільшенням параметра n: від ~150–200 KB при $n = 128$ до понад 8 MB при $n \approx 1000$. Такий характер (наближено квадратичний) узгоджується з теоретичною оцінкою, оскільки публічний ключ містить матрицю A розміру $m \times n$ та вектор b, а в експерименті покладено $m \approx n$, тобто $| pk | \sim O(n2)$. Практичний висновок полягає в тому, що збільшення розмірності для підвищення криптостійкості дуже швидко робить схему затратною за пам’яттю і пропускною здатністю: передавання та зберігання відкритого ключа для великих n стає основним вузьким місцем, тому при проєктуванні LWE-криптосистем необхідно явно враховувати компроміс між рівнем безпеки та допустимим обсягом інформації про ключі.

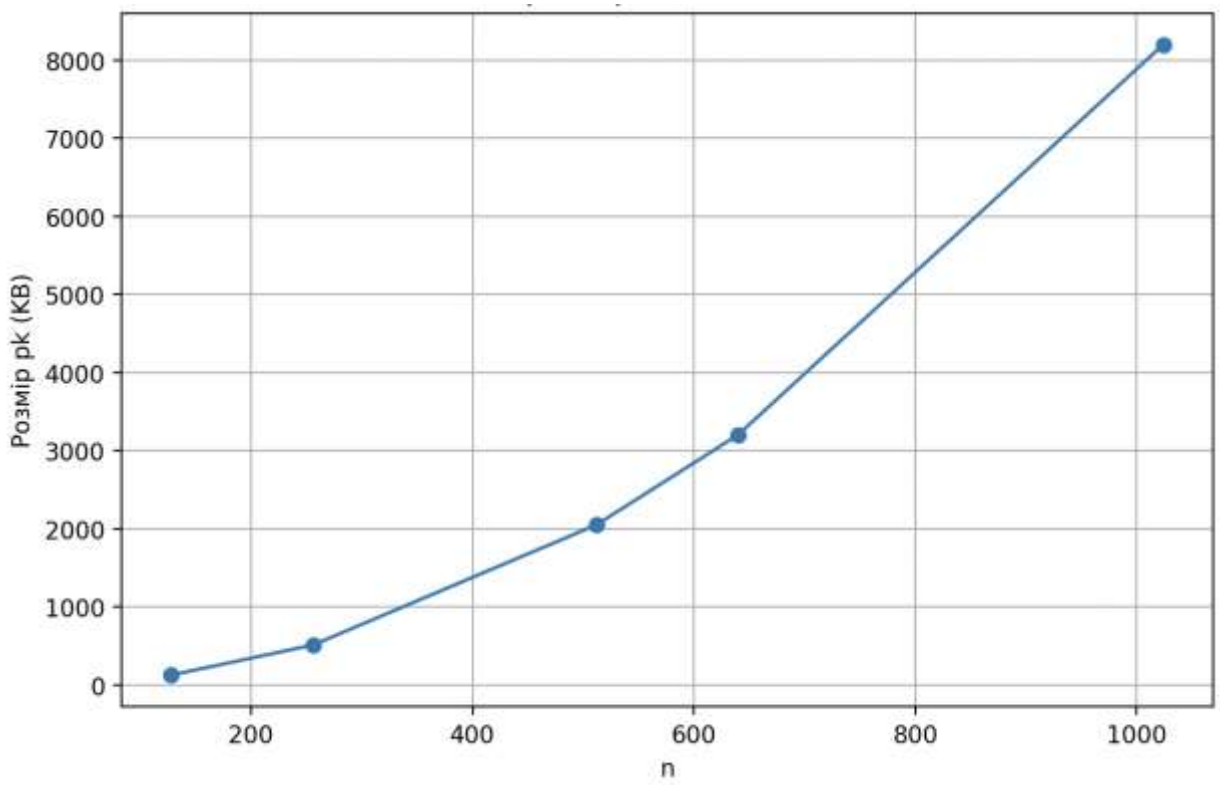

Рисунок 3.2 – Залежність розміру відкритого ключа від n

Експеримент 2. Вплив кількості зразків m на продуктивність і ресурси

При фіксованих n, q змінюємо кількість LWE-зразків m (рядків матриці A) і дивитися, як це впливає на часу шифрування й розмір відкритого ключа.

2.1. План експерименту.

Фіксуємо, наприклад:

- $n = 640; q = 215; \alpha = 1.0$.

Вибираємо:

$m \in \{n, 2n, 4n\}$ (або більше варіантів: $0.5n,\ n, 1.5n, 2n$).

Для кожного m генеруємо ключі.

Вимірюємо:

– час KeyGen (генерація A, b);

– час Enc (оскільки множення $A^T x$ має складність $O(n\ m)$);

– розмір відкритого ключа (кількість байт).

2.2. Аналізуємо:

- розмір pk, який росте лінійно з m;

– час шифрування теж приблизно лінійний по m.

Отже, більший m – потенційно більше зразків LWE (може впливати на безпеку в бік послаблення, якщо атакувальник має дуже багато відповідей).

При цьому для базової PKE/KEM часто достатньо $m \approx n$ або трохи більше.

Рисунок 3.3 показує, що при фіксованому значенні n збільшення кількості LWE-зразків m призводить до майже лінійного зростання часу шифрування, тоді як час дешифрування практично не змінюється й залишається наближено нульовим на обраному масштабі. Це відповідає теоретичним оцінкам: процедура шифрування містить множення матриці A розміру $m \times n$ на вектор, тобто має складність $O(m\,n)$, тоді як дешифрування виконує лише один скалярний добуток $\langle u, s \rangle$ довжини n, незалежний від m.

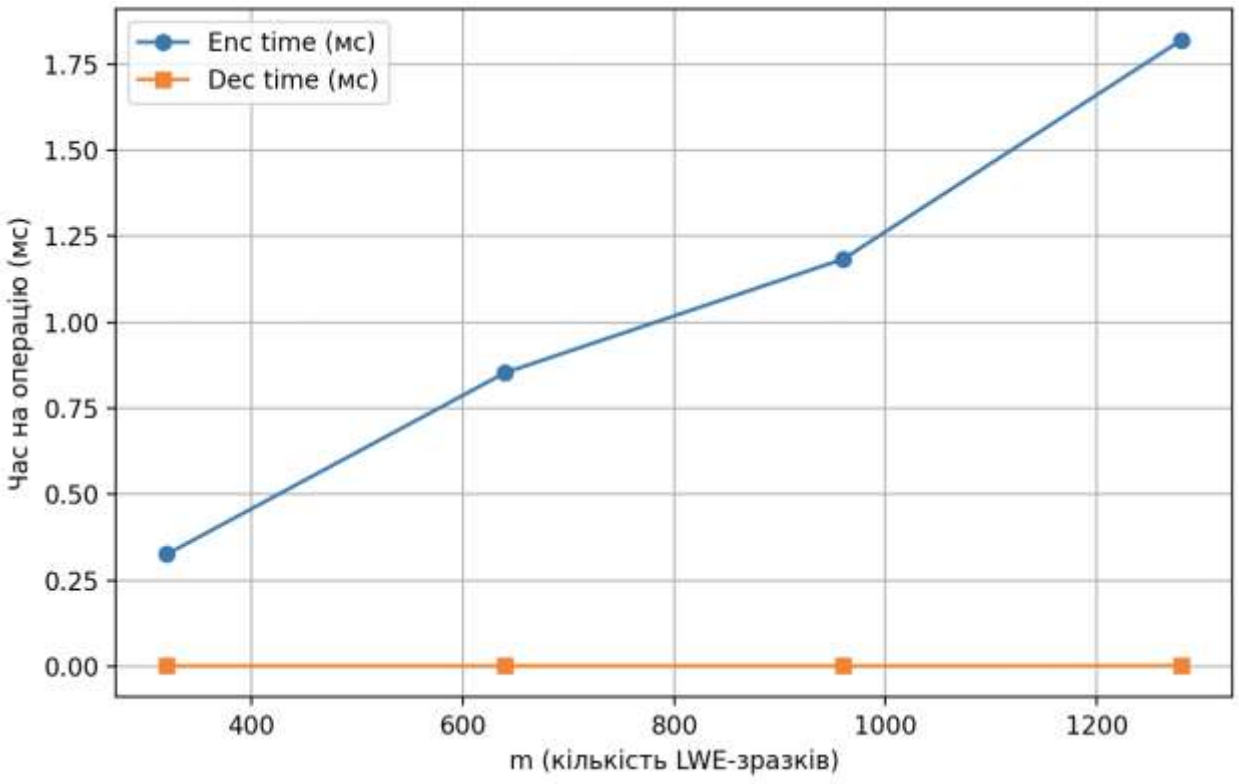

Рисунок 3.3 – Залежність часу шифрування/дешифрування від m (n фіксоване)

Отже, параметр m суттєво впливає на продуктивність сторони, що шифрує (особливо сервера/клієнта, який генерує багато шифротекстів), тоді як вартість дешифрування залишається стабільною, що важливо при проєктуванні протоколів із ресурсно обмеженими отримувачами (рис.3.4).

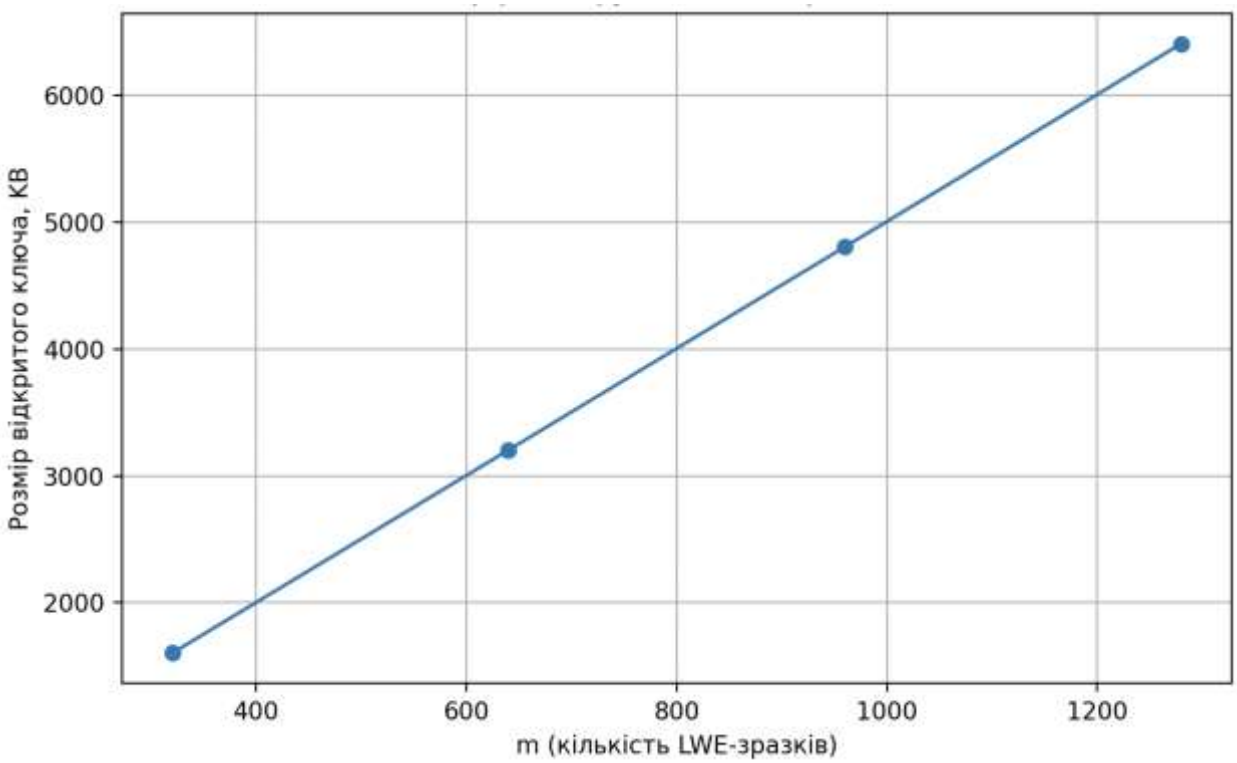

Рисунок 3.4 –Залежність розміру pk від m (n фіксоване)

Рисунок 3.5 демонструє, що час шифрування LWE-схеми з фіксованим n зростає майже лінійно зі збільшенням коефіцієнта factor (тобто кількості зразків $m = factor \cdot n$): при переході від 0.5·n до 2·n середній час шифрування збільшується приблизно з 0.33 мс до 1.82 мс. Такий характер залежності добре узгоджується з теоретичною складністю операції шифрування $O(m\,n)$: за фіксованого n вона фактично стає лінійною за m. Отже, параметр m є прямим «регулятором» вартості шифрування: збільшення кількості рядків матриці A для потенційного посилення стійкості або гнучкості протоколу безпосередньо приводить до пропорційного погіршення продуктивності сторони, що шифрує.

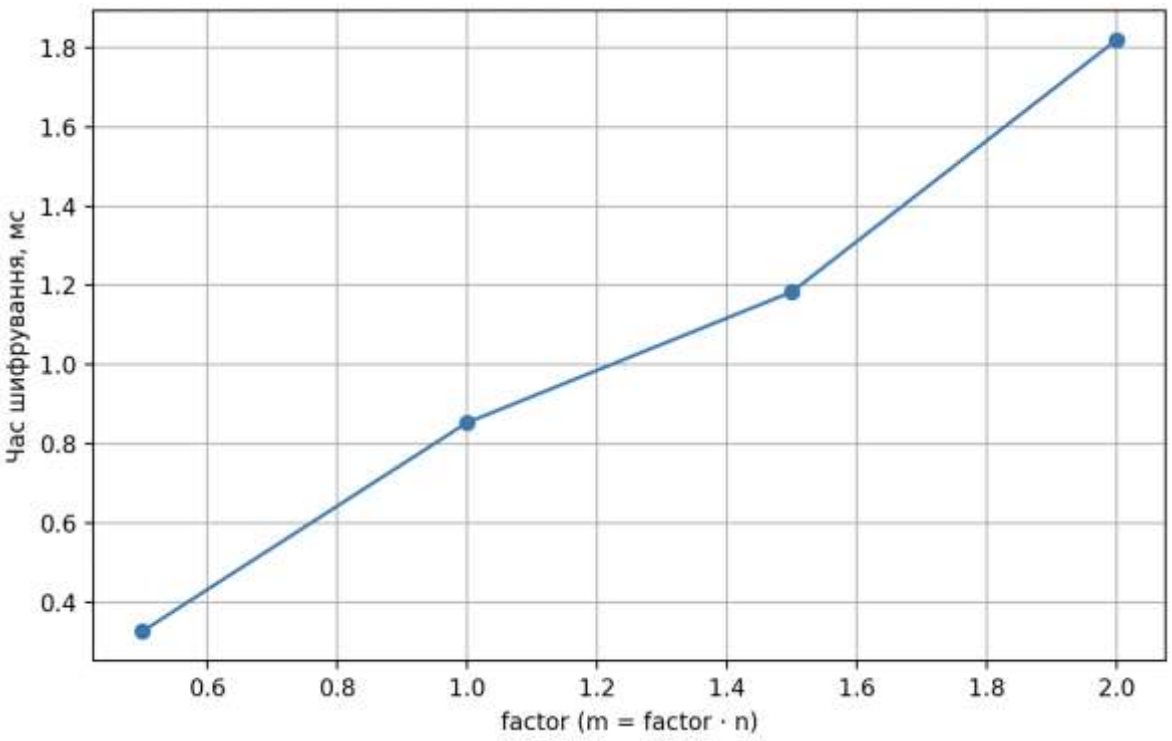

Рисунок 3.5 –Залежність часу шифрування від множника m (при фіксованому n)

Експеримент 3. Вплив модуля q на швидкодію та коректність дешифрування

При фіксованих n, m змінюємо модуль q і аналізуємо:

- як змінюється час модулярних операцій;
- як змінюється «запас по шуму» і ймовірність помилки дешифрування.

1. План експерименту

Фіксуємо:

- $n = 640, m \approx n$;
- один і той самий спосіб генерування шуму (наприклад дискретний гаус з фіксованим σ, або простий $\{-1, 0, 1\}$ для тестів).

Візьмемо: $q \in \{2^{12}, 2^{13}, 2^{14}, 2^{15}, 2^{16}\}$ або кілька простих чисел зі схожими бітовими розмірами.

Для кожного q:

1. Згенерувати ключі.

2. Провести N експериментів «шифрування + дешифрування» випадкових бітів (наприклад 100 000).

3. Для кожного шифрування записувати:

- час шифрування /розшифрування (Enc/Dec);
- чи правильно дешифрування відновило біт.

4. Обчислити:

- середній час Enc/Dec;
- емпіричну ймовірність помилки дешифрування $P_{fail}(q)$.

.2. Провести дослідження:

- $T_{Enc}(q), T_{Dec}(q)$ від $log_2\ q$ (час має зростати, бо модулярні операції складніші);
- $P_{fail}(q)$: для малих q очікувано вищий відсоток помилок (шум частіше «перестрибує» межу).

Як видно з рисунку 3.6 при заданих значеннях ймовірність помилки не залежить від модуля q.

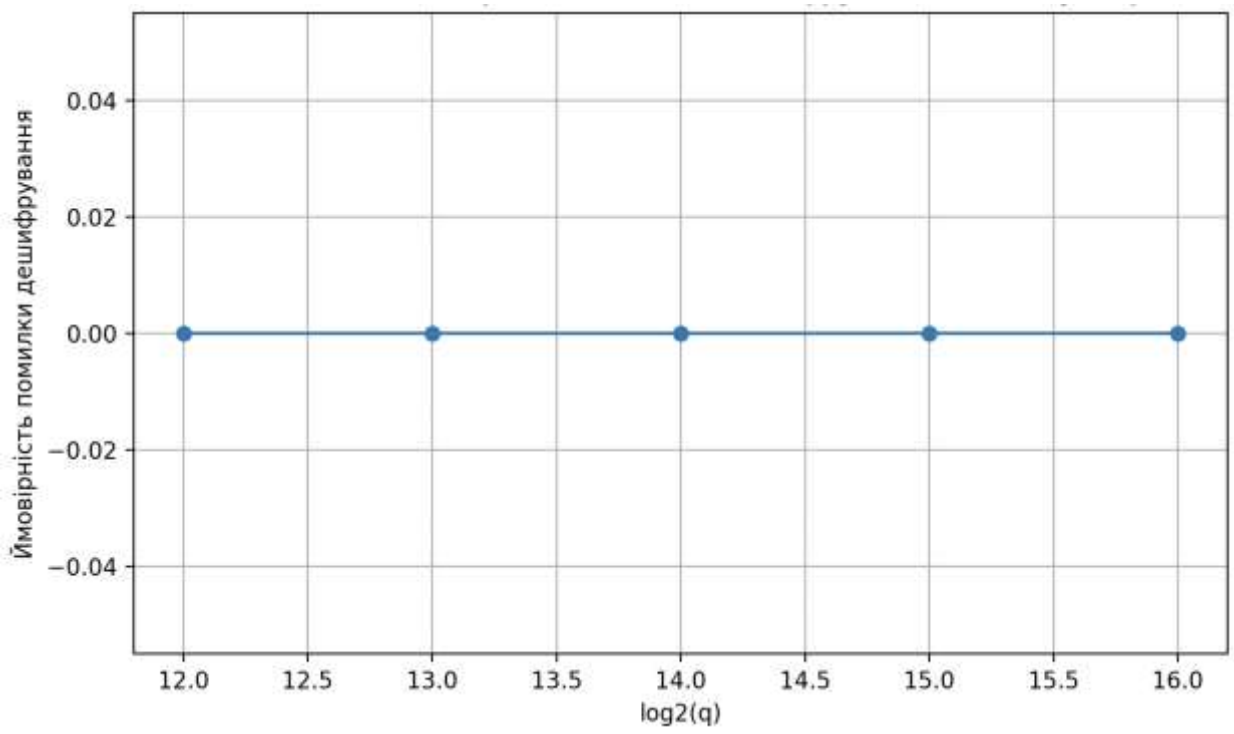

Рисунок 3.6 – Залежність ймовірності помилки дешифрування від модуля q

Рисунок 3.7 показує, що зміна модуля q у діапазоні $log_2\ (q) \in [12, 16]$ практично не впливає на час шифрування й дешифрування: середній час шифрування (Enc) коливається в межах $\sim 0.68 - 0.74$ мс, а розшифрування (Dec) залишається близьким до нуля на обраному масштабі, причому спостережувані відхилення мають характер скоріше вимірювального шуму, ніж систематичної тенденції. Це узгоджується з тим, що основну частку обчислювальних витрат становить множення матриця–вектор розміру $m \times n$, яке не залежить від числового значення модуля (при використанні однакової машинної розрядності, наприклад 64-бітних цілих), а зміна q впливає лише на константні витрати модулярної редукції, які на практиці виявляються незначними.

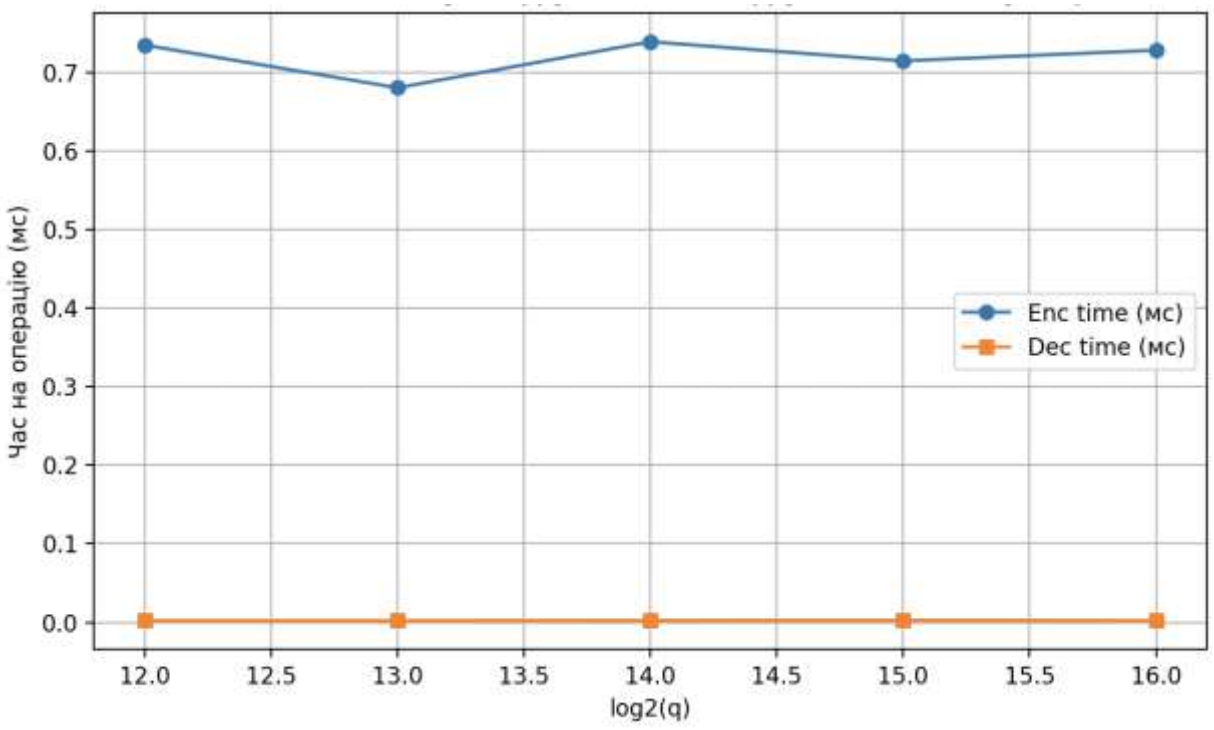

Рисунок 3.7 – Залежність часу шифрування/дешифрування від модуля q

Таким чином, вибір модуля в розумному діапазоні бітової довжини визначається переважно вимогами до безпеки та стійкості до шуму, а не впливом на продуктивність.

Експеримент 4. Комплексне порівняння наборів параметрів

Вибираємо 3 реалістичні набори параметрів, прив'язані до різних рівнів безпеки (умовно «рівень 128 біт», «192 біт», «256 біт»), і порівнюємо їх по продуктивності.

4.1. План експерименту

Вибираємо три профілі:

профіль A (≈128 біт):

$$n1 = 640, q1 = 2^{15}.$$

профіль B (≈192 біт):

$$n2 = 976, q2 = 2^{15}.$$

профіль C (≈256 біт):

$$n3 = 1344, q3 = 2^{16}.$$

Це можна прямо позначити як «аналог Frodo-640/976/1344», без повного копіювання їхніх параметрів.)

Для кожного профілю:

1. Заміри часу KeyGen/Enc/Dec (по багаторазових запусків).
2. Заміри розмірів ключів і шифротекстів.
3. Тестування ймовірності помилки дешифрування.

4.2. Аналізуємо «профіль – час – розміри – орієнтовний рівень безпеки».

Результати досліджень відображаємо на графіках:

- залежність часу від орієнтовного рівня безпеки (рис.3.8);
- залежність розміру відкритого ключа від рівня безпеки (рис.3.9).

Час шифрування/дешифрування залежно від рівня безпеки (по осі X – security_level = 128/192/256 біт, по осі Y – T_enc_ms та T_dec_ms).

Видно, що час шифрування зростає майже лінійно з рівнем безпеки (≈0.8 → 2.1 → 5.0 ms); час дешифрування – на порядки менший і майже не змінюється (рис.3.8).

Розмір відкритого ключа залежно від рівня безпеки (по осі X – security_level, по Y – size_pk_KB). Тут чітко видно ріст: ~3205 KB → ~7449 KB → ~14122 KB при переході 128 → 192 → 256 біт (рис.3.9).

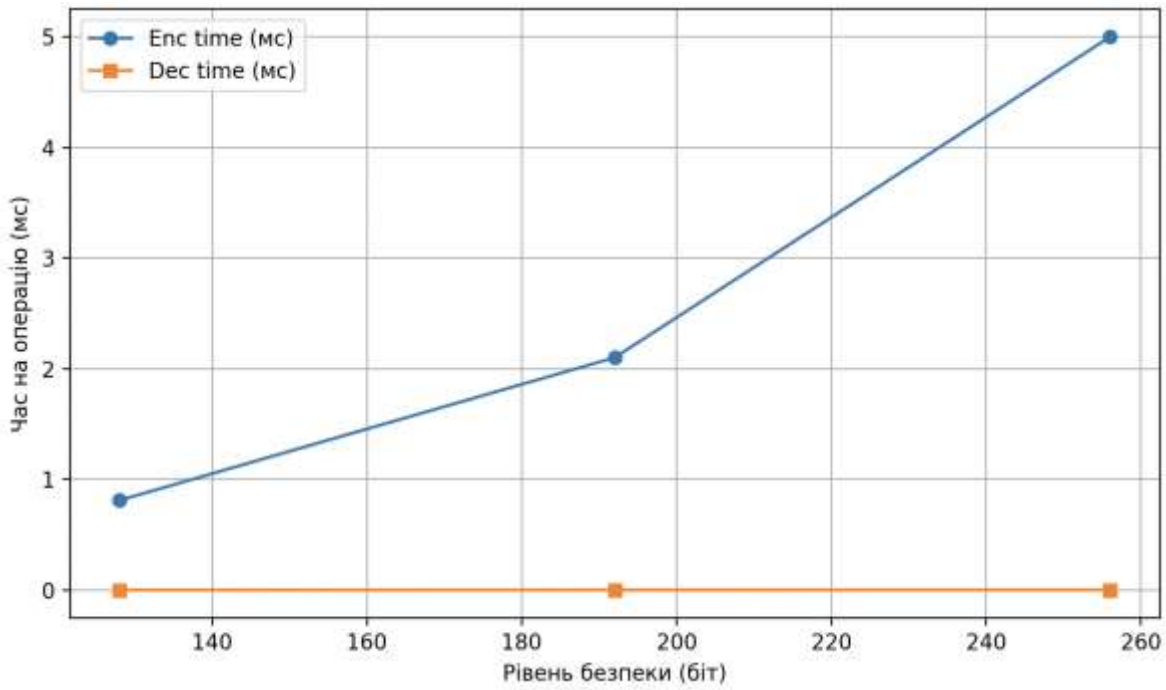

Рисунок 3. 8 – Час шифрування/дешифрування залежно від рівня безпеки

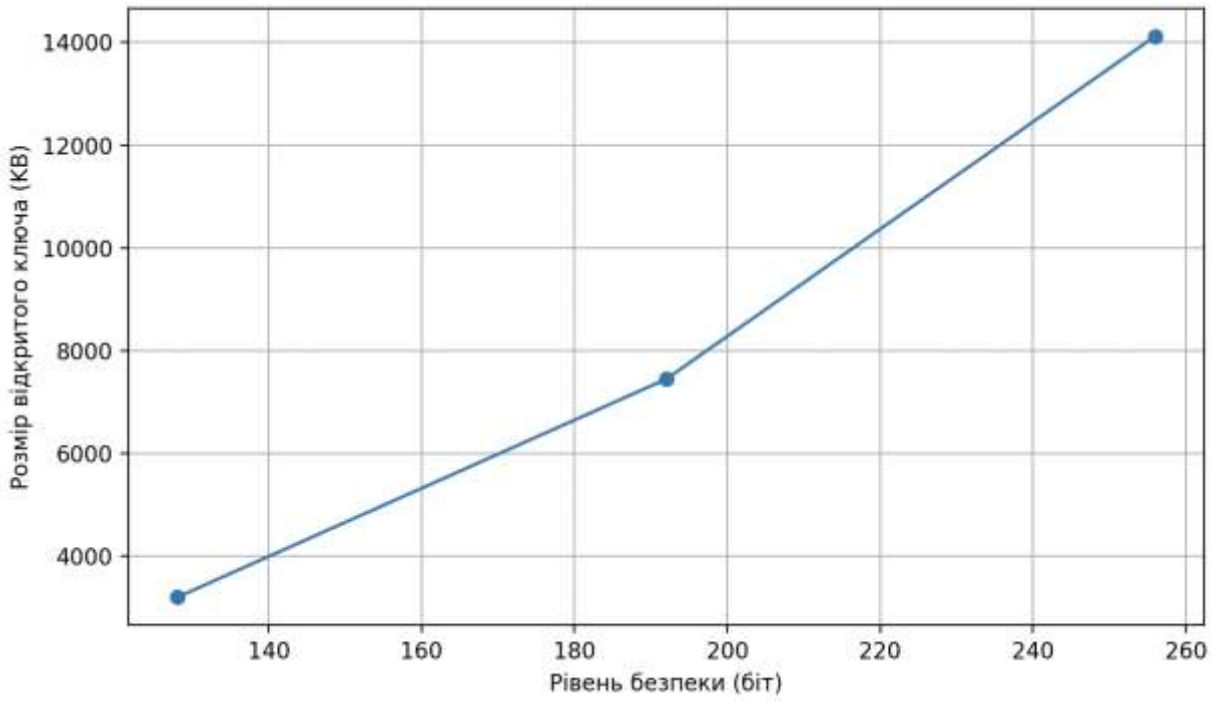

Рисунок 3.9 – Розмір відкритого ключа залежно від рівня безпеки

3.3 Вплив параметрів на глибину гомоморфних обчислень

Суть експерименту дослідити, як обрані параметри впливають на максимальну глибину обчислень, при якій дешифрування ще коректне (до того, як шум «зламає» повідомлення).

1. План експерименту.

Зафіксувати кілька наборів (n, m, q) (наприклад, ті ж профілі A,B,C).

Для кожного профілю:

– зашифрувати L випадкових бітів/чисел;

– послідовно виконати k разів гомоморфне додавання (або інші прості операції, які збільшують шум).

Після кожного кроку:

- виконати дешифрування;
- зафіксувати, чи правильний результат (0/1);
- оцінити «відстань до межі» (напр., $| w |$ або відстань до $\lfloor q/2 \rfloor$).

2. Дослідити залежність «максимальна глибина $kmax$, при якій помилок немає» від параметрів (n, m, q).

Порівняння профілів: чи дає більший модуль q або більший n помітний виграш у допустимій глибині гомоморфних обчислень.

Встановлюємо параметри:

1) беремо параметри LWE (як у попередніх експериментах);
2) фіксуємо ключі (pk, sk);
3) для глибини d:

– шифруємо випадковий біт μ;

– d разів додаємо до нього шифротекст від нуля: c←c⊕Enc(0), це збільшує шум, але відкритий текст має залишатися μ;

– дешифруємо й перевіряємо, чи μdec=μ;

– для кожної глибини рахуємо частку помилок дешифрування → failure_rate(depth) (рис.3.10).

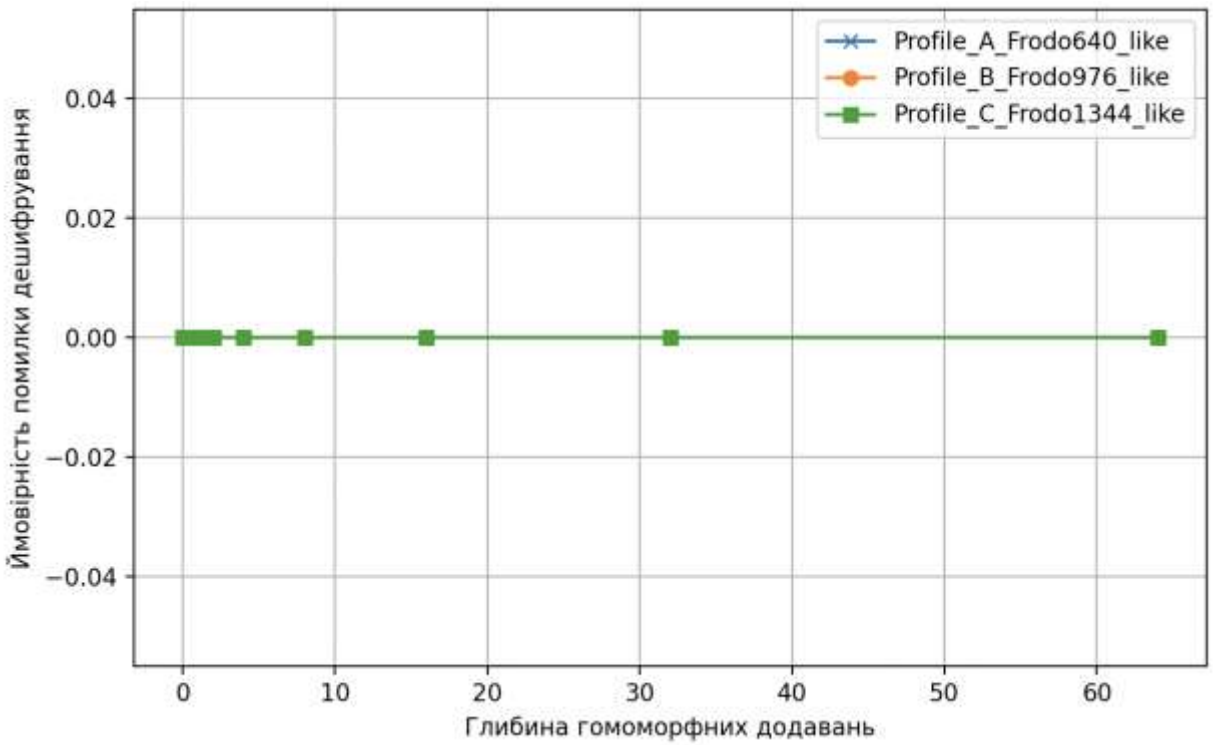

Рисунок 3.10 –Залежність ймовірності помилки від глибини гомоморфних обчислень

За результатами експерименту (рис. 3.11) видно, що середній час виконання одного тесту зростає майже лінійно зі збільшенням глибини гомоморфних додавань, причому крутість нахилу залежить від «ваги» параметрів схеми: профіль C ($n = 1344, q = 2^{16}$) демонструє найвищі затрати (≈300 мс при depth=64), профіль B ($n = 976$) – проміжні значення (≈110 мс), а профіль A ($n = 640$) – найменший час (≈45 мс).

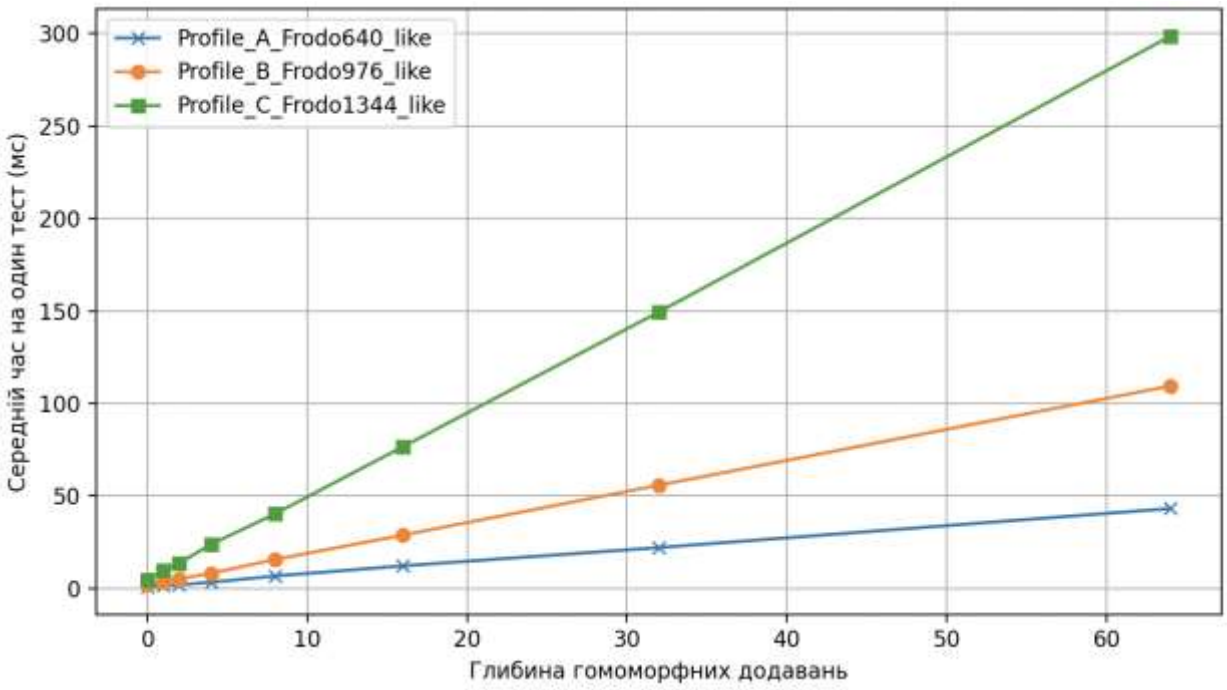

Рисунок 3.11 – Залежність часу обчислень від глибини гомоморфного додавання

Це підтверджує, що збільшення розмірності параметрів, яке відповідає підвищенню рівня криптостійкості, лінійно «збільшується» з точки зору часу гомоморфних обчислень: кожне додаткове гомоморфне додавання в профілі C потребує у кілька разів більше часу, ніж у профілі A, тому при проєктуванні LWE-гомоморфних алгоритмів необхідно балансувати між цільовим рівнем безпеки та допустимою глибиною обчислень і продуктивністю.

ВИСНОВКИ

В кваліфікаційній роботі розв’язано актуальну задачу підвищення ефективності алгоритмів гомоморфного шифрування на основі задачі навчання з помилками. При цьому отримано наступні результати.

1. У роботі систематизовано теоретичні основи гомоморфного шифрування: проаналізовано часткові та повні схеми, їхню алгоритмічну складність, продуктивність та відомі обмеження. Показано, що практичне використання FHE-схем досі суттєво обмежене часовими та обчислювальними витратами.

2. На основі задачі навчання з помилками розроблено та формально описано схему гомоморфного шифрування з підтримкою гомоморфного додавання. Сформульовано постановку задачі, визначено формат ключів, шифротекстів та операції розшифрування.

3. Запропоновано та реалізовано підхід до оптимізації модулярних обчислень у LWE-орієнтованій схемі, що дає змогу зменшити часові витрати базових операцій без погіршення коректності розшифрування. Показано, що використання ефективних методів роботи за модулем є основою для підвищення продуктивності гомоморфних перетворень.

4. Проведено порівняльний аналіз сучасних бібліотек та фреймворків гомоморфного шифрування з позицій підтримуваних схем, модель безпеки, продуктивності та зручності інтеграції у прикладні системи. На основі аналізу сформовано узагальнені рекомендації щодо вибору бібліотек для задач із різними вимогами до точності обчислень та глибини гомоморфних операцій.

5. Експериментально досліджено вплив параметрів LWE-схеми (розмір вектора, модуль, дисперсія шуму, кількість вибірок тощо) на продуктивність алгоритму та глибину допустимих гомоморфних обчислень. Встановлено, що збільшення рівня безпеки через зростання розмірності та модуля призводить до суттєвого ускладнення обчислень.

6. На основі проведених досліджень сформульовано практичні рекомендації щодо налаштування параметрів LWE-схем для прикладних сценаріїв із обмеженою глибиною гомоморфних обчислень. Показано, що правильно підібрані параметри дозволяють досягти прийнятного балансу між безпекою, продуктивністю та точністю результатів, що підтверджує доцільність застосування гомоморфного шифрування на основі навчання з помилками в реальних інформаційних системах.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Acar, A., Aksu, H., Uluagac, A. S., & Conti, M. (2018). A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys, 51(4), 79:1–79:35. https://doi.org/10.1145/3214303

2. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., … Vaikuntanathan, V. (2018). Homomorphic Encryption Standard *(v1.1)*. HomomorphicEncryption.org. Retrieved from https://homomorphicencryption.org

3. Homomorphic Encryption with Images. Режим доступу: https://azeemba.com/posts/homomorphic-encryption-with-images.html

4. Boneh, D., Goh, E. J., & Nissim, K. (2005). Evaluating 2-DNF formulas on ciphertexts. In J. Kilian (Ed.), Theory of Cryptography (TCC 2005) (Lecture Notes in Computer Science, Vol. 3378, pp. 325–341). Springer. https://doi.org/10.1007/978-3-540-30576-7_18

5. Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017). Homomorphic encryption for arithmetic of approximate numbers. In T. Takagi & T. Peyrin (Eds.), *Advances in Cryptology – ASIACRYPT 2017* (Lecture Notes in Computer Science, Vol. 10624, pp. 409–437). Springer. https://doi.org/10.1007/978-3-319-70694-8_15

6. Chillotti, I., Gama, N., Georgieva, M., & Izabachène, M. (2020). TFHE: Fast fully homomorphic encryption over the torus. Journal of Cryptology, 33(1), 34–91. https://doi.org/10.1007/s00145-019-09319-x

7. Ducas, L., & Micciancio, D. (2015). FHEW: Bootstrapping homomorphic encryption in less than a second. In E. Oswald & M. Fischlin (Eds.), *Advances in Cryptology – EUROCRYPT 2015* (Lecture Notes in Computer Science, Vol. 9056, pp. 617–640). Springer. https://doi.org/10.1007/978-3-662-46800-5_24

8. Fan, J., & Vercauteren, F. (2012). Somewhat practical fully homomorphic encryption. IACR Cryptology ePrint Archive, 2012/144. Retrieved from https://eprint.iacr.org/2012/144

9. Ferrara, M., Tortora, A., & Tota, M. (2025). An overview of torus fully homomorphic encryption. International Journal of Group Theory, 14(2), 59–73. https://doi.org/10.22108/ijgt.2023.139030.1869

10. Gentry, C. (2009). Fully homomorphic encryption using ideal lattices. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing (STOC '09)* (pp. 169–178). ACM. https://doi.org/10.1145/1536414.1536440

11. Li, B., & Micciancio, D. (2021). On the security of homomorphic encryption on approximate numbers. In A. Canteaut & F.-X. Standaert (Eds.), *Advances in Cryptology – EUROCRYPT 2021* (Lecture Notes in Computer Science, Vol. 12696, pp. 648–677). Springer. https://doi.org/10.1007/978-3-030-77870-5_23

12. Paillier, P. (1999). Public-key cryptosystems based on composite degree residuosity classes. In J. Stern (Ed.), *Advances in Cryptology – EUROCRYPT '99* (Lecture Notes in Computer Science, Vol. 1592, pp. 223–238). Springer. https://doi.org/10.1007/3-540-48910-X_16

13. Thaine, P. (2018, December 26). *Homomorphic encryption for beginners: A practical guide (Part 1)*. Medium. Retrieved from https://medium.com/privacy-preserving-natural-language-processing/homomorphic-encryption-for-beginners-a-practical-guide-part-1-b8f26d03a98a

14. Zama: Homomorphic Encryption 101. Режим доступу: https://www.zama.org/post/homomorphic-encryption-101

15. Splunk: Homomorphic Encryption: How It Works. Режим доступу: https://www.splunk.com/en_us/blog/learn/homomorphic-encryption.html

16. GeeksforGeeks: Homomorphic Encryption. Режим доступу: https://www.geeksforgeeks.org/ethical-hacking/homomorphic-encryption/

17. Li, J., Qiao, Z., Zhang, K., & Cui, C. (2021). A lattice-based homomorphic proxy re-encryption scheme with strong anti-collusion for cloud computing. *Sensors*, *21*(1), 288.

18. Regev, O. (2009). On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM), 56(6), 1-40.

19. Full Homomorphic encryption. Режим доступу: https://fhe.org/resources/

20. Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017, November). Homomorphic encryption for arithmetic of approximate numbers. In *International conference on the theory and application of cryptology and information security*, pp. 409-437

21. Marcolla, C., Sucasas, V., Manzano, M., Bassoli, R., Fitzek, F. H., & Aaraj, N. (2022). Survey on fully homomorphic encryption, theory, and applications. *Proceedings of the IEEE*, *110*(10), pp. 1572-1609.

22. Martins, P., Sousa, L., & Mariano, A. (2017). A survey on fully homomorphic encryption: An engineering perspective. *ACM Computing Surveys (CSUR)*, *50*(6), 1-33.

23. Moore, C., O'Neill, M., O'Sullivan, E., Doröz, Y., & Sunar, B. (2014, June). Practical homomorphic encryption: A survey. In *2014 IEEE International Symposium on Circuits and Systems (ISCAS)*б pp. 2792-2795)

24. Doan, T. V. T., Messai, M. L., Gavin, G., & Darmont, J. (2023). A survey on implementations of homomorphic encryption schemes. *The Journal of Supercomputing*, *79*, 15098-15139.

25. Alloghani, M., Alani, M. M., Al-Jumeily, D., Baker, T., Mustafina, J., Hussain, A., & Aljaaf, A. J. (2019). A systematic review on the status and progress of homomorphic encryption technologies. *Journal of Information Security and Applications*, *48*, 102362.

26. Microsoft SEAL. Режим доступу: https://www.microsoft.com/en-us/research/project/microsoft-seal/#:~:text=Microsoft%20SEAL—powered%20by%20open,their%20key%20with%20the%20service

27. FHE Libraries: Established Cryptographic Building Blocks. https://el3ctrum.com/uncategorized/fhe-libraries-established-cryptographic-building-blocks/#:~:text=Background%20%26%20Developers%3A%20HEAAN%20stands,the%20library's%20performance%20and%20features

28. OpenFHE. Режим доступу: https://openfhe.org

29. Private Join and Compute. Режим доступу: https://github.com/google/private-join-and-compute

30. Yuan, J., Liu, W., Shi, J., & Li, Q. (2025). Approximate homomorphic encryption based privacy-preserving machine learning: A survey. *Artificial Intelligence Review, 58*(3), 82. https://doi.org/10.1007/s10462-024-11076-8

31. Wu, L., Wang, X. A., Liu, J., Su, Y., Tu, Z., Liu, W., … Zhang, J. (2025). Homomorphic encryption for machine learning applications with CKKS algorithms: A survey of developments and applications. *Computers, Materials & Continua, 85*(1), 89–119. https://doi.org/10.32604/cmc.2025.064346

32. Ezeogu, A. O. (2025). Homomorphic encryption in healthcare analytics: Enabling secure cloud-based population health computations. *Journal of Advanced Research, 1*(2), 42–60.

33. Ilyenko, A. (2023). Practical aspects of using fully homomorphic encryption systems to protect cloud computing. In *Proceedings of the 3rd International Workshop on Cybersecurity* (pp. 225–229). CEUR-WS. http://ceur-ws.org/Vol-3550/short5.pdf

34. Jorge, H., Silva, J., & Ferreira, A. (2025). Evaluating homomorphic encryption schemes for privacy-preserving analysis of healthcare data. *Digital, 5*(3), 74. https://doi.org/10.3390/digital5030074

35. Shen, H., Xu, Q., Yu, B., Yang, Y., & He, W. (2025). Bootstrapping in approximate fully homomorphic encryption: a research survey. Cybersecurity, 8(1), 87.

36. Homomorphic Encryption: What Is It, and Why Does It Matter? Режим доступу: https://www.internetsociety.org/resources/doc/2023/homomorphic-encryption/

37. What is homomorphic encryption, and why isn’t it mainstream? Режим доступу: https://www.keyfactor.com/blog/what-is-homomorphic-encryption/

38. Brakerski, Z., & Vaikuntanathan, V. (2014). Efficient fully homomorphic encryption from (standard) LWE. SIAM Journal on computing, 43(2), 831-871.

39. Barrett reduction. Режим доступу: https://en.wikipedia.org/wiki/Barrett_reduction

40. Montgomery modular multiplication. Режим доступу: https://en.wikipedia.org/wiki/Montgomery_modular_multiplication

41. Басістий В.П., Логош В.Д. Алгоритми гомоморфного шифрування. Матеріали X Міжнародної науково-технічної конференції «Інформаційно-комп’ютерні технології», м. Житомир, 28-29 березня 2025 р. – Житомир: Житомирська політехніка, 2025. – С.129-130.

42. Логош В, Смірнов Д., Хомяк Р. Популярні бібліотеки та фреймворки гомоморфного шифрування. Матеріали проблемно-наукової міжгалузевої конференції «Кібербезпека та комп’ютерно-інтегровані технології» (КБКІТ - 2025), Тернопіль, 2025. – С. 93-95.

ДОДАТОК А

Код програми гомоморфного арифметичного додавання

```
import numpy as np
import random

# Параметри  RLWE-схеми

# Кільце R_q = Z_q[x]/(x^N + 1), N - степінь, q - модуль
N = 8                # степінь полінома (малий для демо)
Q = 32768            # модуль коефіцієнтів (2^15)
T = 16               # модуль повідомлень (plaintext modulus)
DELTA = Q // T       # масштаб для кодування plaintext

def mod_q(poly, q=Q):
    """Звести всі коефіцієнти полінома по модулю q в [0,q)."""
    return np.mod(poly, q)

def poly_add(a, b, q=Q):
    """Поліноміальне додавання в R_q."""
    return mod_q(a + b, q)

def poly_sub(a, b, q=Q):
    """Поліноміальне віднімання в R_q."""
    return mod_q(a - b, q)

def poly_mul(a, b, q=Q, n=N):
    """
    Поліноміальне множення в R_q = Z_q[x]/(x^n + 1).

    Робимо наївну згортку (O(n^2)), потім редукція по модулю x^n + 1:
      x^n = -1  => член x^{n+k} з коефіцієнтом c додає -c до x^k.
    """
    # звичайна згортка довжини 2n-1
    res_long = np.convolve(a, b)

    # редукція по x^n + 1
    res = np.zeros(n, dtype=int)
    for i, coeff in enumerate(res_long):
        if i < n:
            res[i] += coeff
        else:
            # x^i = x^{n + k} = -x^k, де k = i - n
            k = i - n
            res[k] -= coeff

    return mod_q(res, q)

def sample_small_poly(n=N, bound=1):
    """
```

```
    Випадковий "малий" поліном (секрет/шум):
    коефіцієнти з діапазону [-bound, bound].
    """
    coeffs = np.random.randint(-bound, bound + 1, size=n)
    return mod_q(coeffs)

def encode_int(m, t=T, delta=DELTA):
    """
    Кодування цілого m в поліном (тільки 0-й коефіцієнт != 0).
    plaintext m ∈ Z_t кодуємо як Delta * m в R_q.
    """
    if not (0 <= m < t):
        raise ValueError(f"m={m} повинен бути в [0, {t-1}]")
    poly = np.zeros(N, dtype=int)
    poly[0] = (m * delta) % Q
    return poly

def decode_int(poly, t=T, delta=DELTA, q=Q):
    """
    Декодування цілого з полінома: дивимось на коефіцієнт при x^0.
    Робимо центроване представлення і ділимо на Delta.
    """
    c0 = int(poly[0] % q)
    # центроване представлення в (-q/2, q/2]
    if c0 > q // 2:
        c0 -= q
    m_hat = int(round(c0 / delta)) % t
    return m_hat

# KeyGen, Encrypt, Decrypt

def rlwe_keygen():
    """
    RLWE KeyGen:
      секретний ключ: s (малий поліном)
      публічний ключ: (a, b), a - випадковий, b = -a*s + e
    """
    s = sample_small_poly()       # секрет
    a = np.random.randint(0, Q, size=N)  # публічний a
    e = sample_small_poly()       # шум
    # b = -a*s + e (mod q)
    as_prod = poly_mul(a, s)
    b = poly_sub(e, as_prod)      # e - a*s
    return (a, b), s

def rlwe_encrypt(m, pk, t=T, delta=DELTA):
    """
    BFV-подібне шифрування цілого m ∈ Z_t.
      pk = (a, b)
      вибираємо "малий" r, e1, e2
      c0 = b*r + e1 + encode(m)
      c1 = a*r + e2
```

```
    """
    a, b = pk
    # повідомлення в R_q
    m_poly = encode_int(m, t=t, delta=delta)

    r  = sample_small_poly()
    e1 = sample_small_poly()
    e2 = sample_small_poly()

    br = poly_mul(b, r)
    ar = poly_mul(a, r)

    c0 = poly_add(br, e1)
    c0 = poly_add(c0, m_poly)
    c1 = poly_add(ar, e2)

    return (c0, c1)

def rlwe_decrypt(ct, sk, t=T, delta=DELTA):
    """
    Дешифрування:
      ct = (c0, c1)
      tmp = c0 + c1*s
      далі декодуємо tmp як integer (дивимось на коефіцієнт x^0).
    """
    c0, c1 = ct
    s = sk
    c1s = poly_mul(c1, s)
    tmp = poly_add(c0, c1s)
    m_hat = decode_int(tmp, t=t, delta=delta)
    return m_hat

# Гомоморфні операції

def rlwe_add(ct1, ct2):
    """
    Гомоморфне додавання:
      (c0, c1) + (d0, d1) = (c0+d0, c1+d1).
    """
    c0, c1 = ct1
    d0, d1 = ct2
    return (poly_add(c0, d0), poly_add(c1, d1))

def rlwe_mul_raw(ct1, ct2):
    """
    "Сире" гомоморфне множення без релініаризації:

      ct1 = (c0, c1) ~ Enc(m1)
      ct2 = (d0, d1) ~ Enc(m2)

      Після множення отримуємо триполіномний шифротекст:
        e0 = c0*d0
```

```
        e1 = c0*d1 + c1*d0
        e2 = c1*d1

      Це BFV-базова формула, але шифроване під "секрет" (1, s, s^2).

    Для справжньої BFV-схеми треба релініаризація через eval key.
    Тут для простоти ми просто збережемо (e0,e1,e2) і
    декодуватимемо через s та s^2.
    """
    c0, c1 = ct1
    d0, d1 = ct2

    c0d0 = poly_mul(c0, d0)
    c0d1 = poly_mul(c0, d1)
    c1d0 = poly_mul(c1, d0)
    c1d1 = poly_mul(c1, d1)

    e0 = c0d0
    e1 = poly_add(c0d1, c1d0)
    e2 = c1d1

    return (e0, e1, e2)

def rlwe_mul_decrypt(ct3, sk, t=T, delta=DELTA):
    """
    Дешифрування "сирого" добутку:
      ct3 = (e0, e1, e2)
      потрібен s^2:
        tmp = e0 + e1*s + e2*s^2
    """
    e0, e1, e2 = ct3
    s = sk

    s2 = poly_mul(s, s)        # s^2

    e1s = poly_mul(e1, s)
    e2s2 = poly_mul(e2, s2)

    tmp = poly_add(e0, e1s)
    tmp = poly_add(tmp, e2s2)

    m_hat = decode_int(tmp, t=t, delta=delta)
    return m_hat

# Демонстрація

if __name__ == "__main__":

    # --- Генерація ключів ---

    pk, sk = rlwe_keygen()
    print("Параметри:")
```

```
print(f"  N = {N}, Q = {Q}, T = {T}, DELTA = {DELTA}")
print("Секретний ключ s:", sk)
print()

# --- Випадкові plaintext'и ---
m1 = random.randint(0, T - 1)
m2 = random.randint(0, T - 1)

print("m1 =", m1)
print("m2 =", m2)
print("Очікуємо (m1 + m2) mod T =", (m1 + m2) % T)
print("Очікуємо (m1 * m2) mod T =", (m1 * m2) % T)
print()

# --- Шифрування ---
ct1 = rlwe_encrypt(m1, pk)
ct2 = rlwe_encrypt(m2, pk)

# --- Перевірка базового Decrypt ---
dec1 = rlwe_decrypt(ct1, sk)
dec2 = rlwe_decrypt(ct2, sk)
print("Dec(m1) =", dec1)
print("Dec(m2) =", dec2)
print("Коректність розшифрування:", dec1 == m1, "і", dec2 == m2)
print()

# --- Гомоморфне додавання ---
ct_add = rlwe_add(ct1, ct2)
dec_add = rlwe_decrypt(ct_add, sk)
print("Гомоморфне додавання:")
print("Dec(Enc(m1)+Enc(m2)) =", dec_add)
print("Чи дорівнює (m1+m2) mod T ? ->", dec_add == (m1 + m2) % T)
print()

# --- Гомоморфне множення (без релініаризації) ---
ct_mul3 = rlwe_mul_raw(ct1, ct2)
dec_mul = rlwe_mul_decrypt(ct_mul3, sk)
print("Гомоморфне множення (raw, з s^2):")
print("Dec(Enc(m1)*Enc(m2)) =", dec_mul)
print("Чи дорівнює (m1*m2) mod T ? ->", dec_mul == (m1 * m2) % T)
print()

print("УВАГА: це навчальна реалізація, параметри не є криптостійкими.")
```

ДОДАТОК Б.
Копії публікацій

Міністерство освіти і науки України,
ДНУ «Інститут модернізації змісту освіти»,
Національний технічний університет України «Київський політехнічний інститут ім. Ігоря Сікорського»,
Інститут кібернетики ім. В.М. Глушкова НАН України,
Інститут телекомунікацій і глобального інформаційного простору НАН України,
Інститут цифровізації освіти НАПН України,
Житомирський державний університет імені Івана Франка,
Житомирський військовий інститут імені С.П. Корольова,
Черкаський державний технологічний університет,
Вінницький національний технічний університет,
Опольська політехніка (Республіка Польща),
Варшавський технологічний університет (Республіка Польща),
Технологічний університет Лулео (Королівство Швеція),
Технічний університет (Чеська Республіка),
Технічний університет (Республіка Болгарія),
Університет країни Басків (Королівство Іспанія),
Віденський технічний університет (Республіка Австрія),
ADA University (Азербайджан)

ТЕЗИ ДОПОВІДЕЙ

XV Міжнародної науково-технічної конференції

Інформаційно-комп'ютерні технології

м. Житомир, 28-29 березня 2025 р.

Житомир
2025

УДК 004
Т11

Рекомендовано до друку Вченою радою Державного університету «Житомирська політехніка» (протокол № 6 від 24.03.2025 р.)

Тези XV Міжнародної науково-технічної конференції
Т11 «Інформаційно-комп'ютерні технології», м. Житомир, 28-29 березня 2025 р. – Житомир: Житомирська політехніка, 2025. – 352 с.
ISBN 978-966-683-698-7

Представлено доповіді учасників XV Міжнародної науково-технічної конференції. Наведено аналіз та результати досліджень сучасних проблем інформаційних технологій, математичного моделювання та розробки програмного забезпечення, інформаційних систем, комп'ютерної інженерії та кібербезпеки, цифрової обробки сигналів та зображень, комп'ютерно-інтегрованих технологій, робототехніки та приладобудування, інформаційних технологій в телекомунікаціях та біомедицині, інформаційно-комунікаційних технологій в освіті.

УДК 004

ISBN 978-966-683-698-7

Наукове видання

Тези XV Міжнародної науково-технічної конференції «Інформаційно-комп'ютерні технології», Житомир, 28-29 березня 2025 р.

Відповідальний за випуск В.В. Болотіна

Свідоцтво про внесення до Державного реєстру суб'єктів видавничої справи ДК № 7177 ВІД 04.11.2021 р.

Адреса редакції: Державний університет «Житомирська політехніка», вул. Чуднівська, 103, м.Житомир, 10005

УДК 004.056

Басістий В.П., аспірант,
Логош В.Д., магістрант
Західноукраїнський національний університет

АЛГОРИТМИ ГОМОМОРФНОГО ШИФРУВАННЯ

Алгоритми гомоморфного шифрування дозволяють виконувати обчислення над зашифрованими даними без їх розшифрування, отримуючи результат, який після розшифрування відповідає результату обчислень над відкритими даними. Ця технологія набуває дедалі більшої актуальності завдяки зростанню обсягів даних, потребі в конфіденційності та розвитку хмарних обчислень.

Алгоритм гомоморфного шифрування на основі "навчання з помилками" (Learning With Errors, LWE) є одним із широко досліджуваних підходів у сучасній криптографії, зокрема в контексті постквантової безпеки та гомоморфного шифрування. Він базується на математичній задачі, яка вважається стійкою навіть до атак квантових комп'ютерів.

LWE був запропонований Одедом Регевом у 2005 році і став основою для багатьох гомоморфних схем, включаючи частково гомоморфні (Partially Homomorphic Encryption, PHE) і повністю гомоморфні (Fully Homomorphic Encryption, FHE) системи [1, 2].

Основна ідея підходу "навчання з помилками" полягає в тому, щоб відрізнити випадкові лінійні рівняння від рівнянь, які мають невеликий "шум" (помилку), доданий до них. Ця складність використовується для створення криптосистем, які дозволяють шифрувати дані таким чином, що обчислення над ними (наприклад, додавання чи множення) можливі без розшифрування.

Алгоритми гомоморфного шифрування на основі LWE використовують математичні проблеми LWE для забезпечення безпеки і можливості виконання операцій над зашифрованими даними. Основні принципи роботи таких алгоритмів включають:

1) генерація ключів. Секретний ключ є випадковим вектором s. Відкритий ключ складається з множини пар (a, b), де $b = \langle a, s \rangle + e \bmod q$;

2) шифрування. Для шифрування повідомлення m, вибирається підмножина пар з відкритого ключа і комбінується їх з повідомленням. Результатом є зашифрований текст, який містить приховане повідомлення плюс невелику помилку;

3) розшифрування. Для розшифрування використовується секретний ключ для обчислення скалярного добутку, від якого потім віднімається

зашифрований текст, щоб отримати повідомлення плюс помилку. Якщо помилка достатньо мала, її можна видалити, щоб отримати оригінальне повідомлення;

4) гомоморфні операції. Алгебраїчна структура зашифрованого тексту дозволяє виконувати операції додавання та множення над зашифрованими даними, які перетворюються на відповідні операції над відкритими даними після розшифрування.

Математично, схеми гомоморфного шифрування на основі LWE зазвичай працюють над кільцями поліномів, що дозволяє ефективно представляти та маніпулювати зашифрованими даними. Популярний підхід полягає у використанні кільцевої версії LWE (Ring-LWE або RLWE), яка забезпечує кращу ефективність порівняно з оригінальною проблемою LWE.

Одним з ключових викликів у гомоморфному шифруванні є управління шумом. Кожна гомоморфна операція, особливо множення, збільшує рівень шуму у зашифрованому тексті. Якщо шум стає занадто великим, правильне розшифрування стає неможливим. На основі LWE розроблено ряд алгоритмів гомоморфного шифрування, зокрема [2]:

BGV (Brakerski-Gentry-Vaikuntanathan). Алгоритм підтримує довільну кількість додавань та обмежену кількість множень, з ефективними методами контролю шуму.

BFV (Brakerski/Fan-Vercauteren). Алгоритм оптимізований для ефективного множення, що широко використовується в практичних реалізаціях.

CKKS (Cheon-Kim-Kim-Song). Алгоритм призначений для обчислень з наближеними числами, що дозволяє ефективно виконувати обчислення з наближеними результатами.

GSW (Gentry-Sahai-Waters). Алгоритм використовує матричну версію LWE для досягнення компактності гомоморфного множення.

Базуючись на математичній проблемі LWE, алгоритми гомоморфного шифрування забезпечують безпеку навіть проти квантових обчислень.

Список використаних джерел:

1. Doan, T. V. T., Messai, M. L., Gavin, G., Darmont, J. A survey on implementations of homomorphic encryption schemes. The Journal of Supercomputing. 2023. Vol.79(13). P.15098-15139.

2. Mahato, G. K., Chakraborty, S. K. A comparative review on homomorphic encryption for cloud security. IETE Journal of Research. 2023. Vol. 69(8). P.5124-5133.

ЗАХІДНОУКРАЇНСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ОДЕСЬКА ПОЛІТЕХНІКА»
ГАЛИЦЬКИЙ ФАХОВИЙ КОЛЕДЖ ІМ. В'ЯЧЕСЛАВА ЧОРНОВОЛА

КІБЕРБЕЗПЕКА ТА КОМП'ЮТЕРНО-ІНТЕГРОВАНІ ТЕХНОЛОГІЇ (КБКІТ – 2025)

науково-практична конференція
молодих вчених, аспірантів та студентів

26–28 серпня 2025
Тернопіль

КРИПТОГРАФІЧНІ МЕТОДИ ЗАХИСТУ ІНФОРМАЦІЇ

УДК 004.056

Вадим ЛОГОШ, Дмитро СМІРНОВ, Роман ХОМЯК

Західноукраїнський національний університет

ПОПУЛЯРНІ БІБЛІОТЕКИ ТА ФРЕЙМВОРКИ ГОМОМОРФНОГО ШИФРУВАННЯ

Вступ. Гомоморфне шифрування дозволяє проводити обчислення безпосередньо над зашифрованими даними. Це дає змогу стороннім сервісам, наприклад, хмарним платформам, обробляти конфіденційну інформацію, не розшифровуючи її. Результат обчислень також залишається зашифрованим і доступним лише власнику секретного ключа, що усуває ризик компрометації даних під час обробки.

Гомоморфні криптосистеми поділяються на частково гомоморфні (PHE), обмежено гомоморфні (SHE/LHE) та повністю гомоморфні (FHE) схеми. Частково гомоморфні забезпечують виконання лише одного типу математичної операції над шифротекстами (лише додавання *або* лише множення). Обмежено гомоморфні дозволяють обмежену кількість як додавань, так і множень (наприклад, довільна кількість додавань і лише одне множення), тобто підтримують обидва типи операцій, але з обмеженою глибиною складання операцій.

Повністю гомоморфні шифрування не мають таких обмежень – вони дозволяють виконувати будь-які обчислення (довільні схеми з додаваннями і множеннями) над зашифрованими даними необмежену кількість разів. FHE-системи, по суті, забезпечують можливість реалізувати на шифротекстах довільну функцію (вони є тюринг-повними), тоді як PHE та обмежені SHE – ні.

Мета. Аналіз сучасних бібліотек та фреймворків гомоморфного шифрування та порівняння їх основних характеристик.

1. Аналіз бібліотек та фреймворків гомоморфного шифрування

Сьогодні існує ряд відкритих бібліотек, які реалізують схеми і надають розробникам зручні інструменти для роботи з гомоморфним шифруванням. Нижче проаналізовано найпоширеніші з них та дано коротку характеристику.

1. Microsoft SEAL. Популярна відкрита бібліотека від Microsoft, що підтримує схеми BFV (Brakerski/Fan–Vercaut.) та CKKS(Cheon -Kim et al.). Орієнтована на простоту використання: надає високорівневий API для виконання гомоморфних обчислень, дозволяючи будувати повністю зашифровані сховища даних і сервіси обробки без розкриття ключів [1].

Написана на C++ (доступні обгортки для .NET, Python), оптимізована для швидкодії, має детальну документацію і приклади.

2. PALISADE. Бібліотека з відкритим кодом, розроблена консорціумом за підтримки DARPA. Підтримує кілька гомоморфних схем – BGV, BFV, CKKS, а також бульові TFHE/FHEW, у тому числі в багатокористувацькому (Multiparty) режимі. Відрізняється модульною архітектурою і гнучкістю налаштувань. На основі PALISADE у 2022 р. створено нову об'єднану платформу OpenFHE.

3. Helib. Одна з перших FHE-бібліотек, розроблена IBM (перший випуск – 2013/2014, публічний реліз – 2016 р.). Реалізує схеми BGV і CKKS (додані

пізніше) та підтримує bootstrapping для BGV. Написана на C++ з відкритим кодом, HElib була націлена передусім на дослідників, надаючи гнучкий, хоч і відносно низькорівневий інтерфейс. IBM продовжує вдосконалювати HElib, зокрема оптимізуючи швидкодію.

4. Бібліотека TFHE. Спеціалізована бібліотека для схеми TFHE (Torus FHE). Розроблена командою дослідників (Chillotti, Gama, Georgieva, Izabachène) і вперше опублікована у 2016–2017 рр. Орієнтована на швидкі булеві операції з частим bootstrapping. Забезпечує виконання логічних схем (AND, OR, XOR тощо) над шифрованими бітами за кілька мілісекунд. Реалізована на C++, використовує швидку операцію БПФ над тором і інші оптимізації. Підтримує також багатокористувацький режим. Недоліком є обмеженість до побітових операцій – для роботи з великими числами чи векторами рекомендується комбінувати її з іншими бібліотеками (або використовувати гібридні підходи).

5. HEAAN (HeaAn). Відкрита бібліотека для схеми CKKS, розроблена дослідниками Сеульського національного університету (авторами CKKS). Назва розшифровується як "Homomorphic Encryption for Arithmetic of Approximate Numbers". Перший випуск – 2016 р., згодом підтримку проекту продовжила корейська компанія CryptoLab [3].

HEAAN реалізує всі основні можливості CKKS: гомоморфні додавання, множення, масштабування, пакування/розпакування векторів. В ній однією з перших з'явилася реалізація bootstrapping для CKKS, що дозволяє виконувати необмежену кількість операцій на зашифрованих речових числах. HEAAN оптимізована для високої точності і швидкості обчислень, підтримує GPU-акселерацію для основних операцій над поліномами. Її часто використовують у дослідженнях, пов'язаних з приватними обчисленнями в AI, оскільки вона швидко впроваджує найновіші алгоритмічні вдосконалення CKKS.

6. OpenFHE. новітній фреймворк (перший реліз – липень 2022) для гомоморфного шифрування, який об'єднує напрацювання кількох попередніх бібліотек [4].

OpenFHE розроблено командою експертів з різних установ під егідою організації Duality Technologies, за участі спільноти (проект під патронатом NumFocus). Бібліотека є наступником PALISADE і включає в себе підтримку всіх основних схем: BGV, BFV, CKKS для арифметичних обчислень, а також схем TFHE і FHEW для булевих операцій. OpenFHE від початку спроектована з урахуванням можливості *bootstrapping* для всіх схем (тобто підтримує перезавантаження і для BGV/BFV/CKKS, чого раніше не було «з коробки») [4].

Великі технологічні компанії вкладаються в розвиток FHE, розробляючи інструменти для спрощення його використання. Наприклад, Microsoft випустила бібліотеку SEAL, яка допомагає інтегрувати гомоморфне шифрування у прикладні рішення (вже реалізовані пілотні проекти з повністю зашифрованого аналізу даних у партнерстві з фінтех-компаніями). Google розробила інструмент Private Join and Compute для захищеного спільного аналізу даних, а також FHE Transpiler – компілятор, що перетворює звичайний код на еквівалентні гомоморфні обчислення [5].

IBM активно працює над прискоренням HElib і пропонує хмарні сервіси з підтримкою FHE. Отже, є підстави вважати, що повністю гомоморфне