НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Кафедра математичних методів захисту інформації

Звіт з комп’ютерного практикуму

«Дослідження сучасних алгебраїчних криптосистем»

з кредитного модуля

«Сучасні алгебраїчні криптосистеми»

Виконали студенти

Групи ФІ-52мн

Булигін Олексій

Остаповець Олеся

Київ 2025

Мета: Дослідження особливостей реалізації сучасних алгебраїчних криптосистем на прикладі алгоритму SMAUG з першого туру національного конкурсу з постквантової криптографії в Кореї (KpqC).

Постановка задачі: у межах практикуму необхідно:

1. реалізувати обраний криптографічний алгоритм SMAUG;
2. підтвердити коректність реалізації за допомогою тестів (за наявності – використати офіційні тестові вектори або референсну реалізацію для перевірки);
3. знайти близькі за класом алгоритми та провести порівняльний аналіз швидкодії за різних умов (параметричні набори, модифікації компонентів);
4. оформити звіт.

Опис труднощів:

Основні складнощі теоретичної частини стосувалися розуміння означень у формулах ($\lfloor \cdot \rceil$)

У практичній реалізації найбільше труднощів було пов’язано з узгодженням теорії з кодом: реалізацією операцій у кільці (множення поліномів з редукцією, матрично-векторні добутки, скалярні добутки), а також з керуванням випадковістю.

1. Теоретичний аналіз

1.1 Визначення та структура алгоритму SMAUG

SMAUG — це ефективний постквантовий механізм інкапсуляції ключів (KEM), безпека якого ґрунтується на складності ґраткових задач, і використовує module-LWE (MLWE) та module-LWR (MLWR) як базові задачі складності.

У специфікації SMAUG.CCAKEM побудова визначається двоступенево:

1. вводиться IND-CPA схема відкритоключового шифрування фіксованих повідомлень довжини 32 байти (SMAUG.CPAPKE),
2. поверх неї застосовується (модифікована, як у Kyber) трансформація Fujisaki–Okamoto для отримання IND-CCA2 KEM.

З огляду на проєктні рішення, SMAUG використовує MLWE як основу стійкості ключів, а MLWR — як основу формування шифротексту, що дає змогу зменшити розмір шифротексту та спростити реалізацію шифрування.

У схемі SMAUG модульні параметри обрано як степені двійки, зокрема $q = 2^{10}$та $p = 2^8$, а також ступінь модульного полінома $n = 256 = 2^8$. Такий вибір спрощує реалізацію арифметики в Z_qта Z_p: редукція за модулем і перетворення між модулями ($q \to p$) можуть реалізовуватися через бітові операції (маскування/зсуви) та контрольоване округлення, що зменшує накладні витрати й полегшує написання високопродуктивної та відтворюваної реалізації.

Додатково в SMAUG застосовано **розріджені тернарні** секретні та еферемерні поліноми (з коефіцієнтами з множини $\{-1,0,1\}$і фіксованою вагою Геммінга). Такий вибір зменшує кількість ненульових коефіцієнтів у векторах sта r, що безпосередньо знижує обчислювальну складність множення у кільці та скорочує обсяг даних під час зберігання/передавання. Крім того, розрідженість полегшує побудову верхніх оцінок сумарної похибки (з урахуванням шуму та округлення), що використовується при аналізі коректності дешифрування.

1.2 Алгебраїчна модель та позначення

Алгоритм SMAUG визначається в кільці многочленів

$$R = Z[x]/(x^n + 1),$$

де n— фіксований параметр схеми, а всі операції виконуються з урахуванням тотожності $x^n \equiv -1$. Для модульної арифметики використовується відповідне фактор-кільце

$$R_q = Z_q[x]/(x^n + 1)$$

а також, у частині формування шифротексту, кільце

$$R_p = Z_p[x]/(x^n + 1).$$

У реалізації це означає, що елементи кільця подаються як поліноми степеня менше nз коефіцієнтами, що беруться за модулем qабо p.

Для масштабування рівня стійкості SMAUG використовує модульну розмірність k, відповідно вектори та матриці розглядаються як елементи просторів R_q^kта $R_q^{k\times k}$. Зокрема, матриця $A \in R_q^{k\times k}$є публічним параметром, який відтворюється детерміновано з короткого зерна, а секретні значення схеми є векторами в R^k(або в підмножинах цього простору).

Для опису “малих” поліномів вводиться множина

$$S_\eta = \{f \in R: deg\ f < n, coef\ (f) \in [-\eta, \eta] \cap Z\},$$

яка використовується для обмеження коефіцієнтів секретних та еферемерних поліномів. Далі у специфікації розглядаються конкретні розподіли для семплінгу секретних значень (зокрема, для векторів s та r).

Матрично-векторні та скалярні добутки (наприклад, $A \cdot r$, $b^T \cdot r$, $c_1^T \cdot s$) виконуються над кільцем R_q з подальшим зведенням коефіцієнтів за модулем. Перехід від значень у R_q до значень у R_p здійснюється операцією масштабування з контрольованим округленням, яка в специфікації задається виразами вигляду $\lfloor \frac{p}{q}(\cdot) \rceil$.

1.3 Симетричні примітиви

У схемі SMAUG використовуються симетричні криптографічні примітиви *H, G, XOF* та *KDF*, які забезпечують детерміноване породження параметрів, “прив’язування” випадковості до відкритого ключа й шифротексту, а також формування спільного секрету у KEM.

- Функція *H* застосовується для хешування відкритого ключа та шифротексту;
- функція G використовується для отримання зерна (seed), що визначає випадковість шифрування;
- *XOF* для розгортання зерна під час генерації матриці *A* та допоміжних значень;
- *KDF* застосовується для виведення остаточного спільного ключа фіксованої довжини з проміжних секретних даних.

1.4 Базова схема відкритого ключа SMAUG.CPAPKE та її складові

SMAUG.CPAPKE є IND-CPA стійкою схемою шифрування, на основі якої будується CCAKEM. Її визначають три алгоритми: **генерація ключів**, **шифрування** та **дешифрування**. Обчислення виконуються в модульному кільці R_q для ключів і проміжних значень та в R_p для компонентів

шифротексту, причому перехід $q \to p$реалізується через масштабування та контрольоване округлення.

– **Генерація ключів**

Алгоритм генерації ключів формує відкритий ключ *pk* та секретний ключ *sk* на основі псевдовипадкового зерна. Матриця $A \in R_q^{k \times k}$ відтворюється детерміновано з короткого значення ρ (seed), що зменшує обсяг відкритого ключа без втрати відтворюваності. Секретний вектор $s \in S_\eta^k$генерується як розріджений тернарний, а додатковий шум $e \in R^k$семплюється з дискретного гаусового розподілу. Вектор відкритого ключа $b \in R_q^k$обчислюється як

$$b = -A^T s + e \pmod q,$$

після чого pk містить (ρ, b), а sk містить s. Така конструкція забезпечує математичну відповідність задачі MLWE в частині формування ключів.

– **Шифрування**

Під час шифрування використовується еферемерний розріджений вектор $r \in S_\eta^k$, який визначає одноразову випадковість. Компонента c_1формується як результат множення $A \cdot r$у R_qіз подальшим переходом до R_pчерез масштабування та округлення:

$$\mathbf{c}_1 := \lfloor \frac{p}{q} \mathbf{A}^T \mathbf{r} \rceil \in R_p^k.$$

Друга компонента c_2включає як скалярний добуток $b^T r$, так і внесок повідомлення μ, закодованого через множник $\frac{q}{t}$:

$$c_2 := \lfloor \frac{p}{q} \left(\boldsymbol{b}^T \boldsymbol{r} + \frac{q}{t} \mu \right) \rceil \in R_p.$$

Таким чином, шифрування реалізує MLWR-структуру в частині формування

шифротексту: “важкі” значення в модулі qквантизуються до модуля p, що зменшує розмір шифротексту.

– **Дешифрування**

Дешифрування виконує компенсування внеску секретного вектора sта відновлення повідомлення з урахуванням квантизації. Для шифротексту (c_2)обчислюється величина $c_2 + c_1^T s$, після чого застосовується масштабування $p \to t$та округлення:

$$\mu' := \lfloor \frac{t}{p}(c_2 + \mathbf{c}_1^T \mathbf{s}) \rceil \in R_t.$$

За умови, що сумарна похибка (шум та похибка округлення) не перевищує порогове значення, отримується $\mu' = \mu$.

1.5 Побудова IND-CCA2 SMAUG.CCAKEM

SMAUG.CCAKEM отримується з SMAUG.CPAPKE шляхом застосування перетворення Fujisaki–Okamoto, яке забезпечує CCA-стійкість за рахунок перевірки узгодженості шифротексту при декапсуляції та використання “резервного” секрету у разі невдачі перевірки.

– **Генерація ключів**

Алгоритм CCAKEM.KeyGen запускає CPAPKE.KeyGen для отримання (s), після чого додатково генерує випадкове значення dфіксованої довжини. Секретний ключ KEM задається як $sk = (s, d)$. Значення dвикористовується як fallback-вхід у KDF у випадку некоректного шифротексту.

– Інкапсуляція

Інкапсуляція формує випадкове μфіксованої довжини та використовує його як внутрішнє “насіння” для шифрування. Для запобігання атакам, пов’язаним із повторним використанням випадковості, зерно для еферемерного вектора rотримується як функція від μта хешу відкритого ключа $H(pk)$. Після

формування $ctxt$остаточний спільний ключ визначається через KDFяк функція від μта хешу шифротексту $H(ctxt)$. Такий підхід забезпечує криптографічне “прив’язування” ключа до конкретного шифротексту.

– **Декапсуляція**

Декапсуляція спочатку відновлює μ'через CPAPKE.Decrypt, після чого повторно обчислює $ctxt'$з використанням того ж детермінованого правила породження випадковості. Далі перевіряється рівність $ctxt = ctxt'$. Якщо перевірка успішна, то спільний ключ обчислюється як $KDF(\mu', H(ctxt))$; якщо перевірка неуспішна, то як $KDF(d, H(ctxt))$. У результаті реакція на некоректний шифротекст маскується, що є критичною умовою досягнення IND-CCA2 стійкості.

1.6 Параметри SMAUG

У SMAUG визначено декілька параметричних конфігурацій, кожна з яких відповідає певному цільовому рівню криптостійкості (SMAUG128 / SMAUG192 / SMAUG256).

Фіксовані параметри:

- Ступінь модульного полінома: $n = 256$.
- Модулі: $q = 1024$, $p = 256$.
- Параметр кодування повідомлення: $t = 2$.
- Стандартне відхилення дискретного гаусового розподілу: $\sigma = 1.0625$

Фіксація n, q, p, tта σуніфікує арифметику та формат подання даних у реалізації

Параметри, що відрізняються в залежності від конфігурації:

- Модульна розмірність k – визначає розміри матриць у $R_q^{k\times k}$та векторів у R_q^k.
- Вага Геммінга секрету h_s – характеризує розрідженість секретного вектора s.

- Вага Геммінга еферемерного значення h_r – характеризує розрідженість еферемерного вектора *r*.

Зміна k, h_sта h_rвикористовується для масштабування схеми між рівнями безпеки, впливаючи на обсяг обчислень.

Набір	Цільовий рівень	(n)	(k)	(q)	(p)	(t)	(h_s)	(h_r)	(σ)
SMAUG128	I	256	2	1024	256	2	140	132	1.0625
SMAUG192	III	256	3	1024	256	2	150	147	1.0625
SMAUG256	V	256	5	1024	256	2	145	140	1.0625

1.7 Реалізаційно важливі складові

Для корекної роботи алгоритму та ефективності реалізації SMAUG, повинні виконуватись наступні вимоги:

- Детермінована генерація матриці A з короткого зерна ρ. Це зменшує обсяг відкритого ключа та гарантує однозначне відновлення параметрів у будь-якій реалізації.
- Генерація розріджених тернарних поліномів (HWTh) для sта r. Це зменшує обчислювальні витрати на множення та дозволяє компактне зберігання через подання множиною індексів ненульових коефіцієнтів.
- Семплінг дискретного гаусового шуму для компоненти e. Це є критичним для стійкості й одночасно чутливим з погляду коректної реалізації, зокрема, щодо стабільності й відсутності реалізаційних витоків.
- Операції масштабування та округлення при переході $q \to p$, які формують MLWR-структуру шифротексту. Коректність дешифрування прямо залежить від узгодженого визначення цих операцій у всіх компонентах реалізації.
- Пакування/розпакування елементів R_q, R_pта розріджених поліномів. Коректна серіалізація визначає сумісність тестових векторів та

відповідність специфікації, а також впливає на фактичні розміри ключів і шифротекстів.

2. Безпека

2.1 Криптографічні припущення та цільові рівні стійкості

Схема SMAUG є постквантовим механізмом інкапсуляції ключа (KEM), побудованим на задачах Module-LWE (MLWE) та Module-LWR (MLWR) у модульних решітках. Автори прямо вказують, що IND-CPA стійкість базової PKE-компоненти SMAUG.PKE спирається на складність MLWE та MLWR, а IND-CCA2 стійкість SMAUG.KEM досягається шляхом перетворення Фуджісакі–Окамото (FO) із «квантовими» модифікаціями (tweaks), аналогічними підходам у Kyber та Saber.

Smaug

Конструктивна ідея SMAUG полягає у розмежуванні джерел стійкості:

- «Key security» (стійкість відновлення секрету з відкритого ключа) пов’язується з MLWE;
- «Ciphertext security» (стійкість шифротексту) — з MLWR, що також дозволяє зменшити розмір шифротексту завдяки переходу від модуля qдо меншого модуля pчерез операції округлення.

Smaug

Окремо наголошується, що MLWR вважається не слабшим за MLWE за умови, що секрет не «перевикористовується» надмірно для генерації великої кількості зразків; у SMAUG MLWR-зразки застосовуються саме в контексті шифрування для підвищення ефективності.

2.2 FO-перетворення та захист від атак обраного шифротексту

SMAUG.KEM конструюється на основі SMAUG.PKE за допомогою Fujisaki–Okamoto transform. У специфікації SMAUG вказано, що під час інкапсуляції еферемерний вектор r, детерміновано породжується як функція від

повідомлення μ та хеша відкритого ключа $H(pk)$. Це рішення зазначається як засіб протидії multi-target атакам.

Фактично, досягнення IND-CCA2 у таких KEM-схемах покладається не лише на редукції безпеки, а й на коректну реалізацію «перевірки» шифротексту (re-encryption check) та на відсутність витоків через поведінку при помилках. Це узгоджується з вимогою, що реакція на некоректний шифротекст має бути замаскована (наприклад, через однаковий шлях обчислень і використання запасного секрету), що є критично для IND-CCA2.

2.3 Декриптаційні помилки

Через використання масштабування й округлення між модулями (перехід $q \to p$) базова PKE-схема може мати ненульову **ймовірність помилки дешифрування**. У статті наведено формалізацію цієї події через норму $\|\cdot\|_\infty$та похибки округлення e_1, e_2, а також оцінку, що ймовірність помилки дешифрування не перевищує δ, де

$$\delta = \Pr\left[\| r^T e + s^T e_1 + e_2 \|_\infty > q/(2t)\right] \quad .$$

Для запропонованих параметричних наборів автори наводять оцінки DFP у логарифмічній шкалі (основа 2), що вказує на малу ймовірність помилки порядку 2^{-160}і менше залежно від набору параметрів, якою можна знехтувати.

З практичної точки зору, контроль DFP важливий не лише для коректності: у багатьох решіткових схемах витоки інформації про факт помилки можуть бути використані в реакційних/побічних атаках. Тому в реалізації KEM критично забезпечити ідентину поведінку для валідних і невалідних шифротекстів.

2.4 Класи релевантних атак на SMAUG

У контексті SMAUG доцільно розглядати такі групи атак з урахуванням підстав безпеки MLWE/MLWR та FO-перетворення:

- Редукційні атаки на MLWE/MLWR:
  Типовий підхід полягає у зведенні задачі до знаходження короткого вектора, SVP/uSVP або близького вектора, CVP/Approx-CVP у відповідній решітці та застосуванні редукції базису, BKZ/DBKZ. Для оцінювання практичної складності часто використовують GSA (Geometric Series Assumption) і модель для BKZ/DBKZ, що описує поведінку норм векторів Грама–Шмідта в «приведеному» базисі.
- Гібридні атаки і вплив розрідженого секрету:
  SMAUG використовує розріджені секрети, що корисно для швидкодії, однак автори окремо відзначають необхідність подальшого вивчення саме sparse LWE/LWR варіантів, оскільки гібридні атаки можуть бути особливо сильними для розрідженого секрету.
- Атаки обраного шифротексту (CCA) на рівні KEM та «multi-target» сценарії:
  Формально IND-CCA2 забезпечується FO-перетворенням; у SMAUG додатково використовується хеш відкритого ключа при генерації випадковості, що прямо позиціонується як засіб протидії multi-target атакам.
- Побічні канали та реалізаційні вразливості
  Стандартні доказові моделі MLWE/MLWR + FO не закривають витоки через час виконання, доступи до пам’яті, помилки округлення/семплінгу тощо. Автори прямо зазначають потребу в подальшому формальному аналізі, включно з атаками на побічні канали, та в «secure implementation against it».

Можна зробити висновок, що заявлена стійкість SMAUG у стандартних моделях зводиться до складності MLWE/MLWR для модульних решіток і коректного застосування FO-перетворення.

Практично релевантний ризик-фактор формується трьома напрямами:

1. ефективність редукційних атак BKZ/DBKZ для конкретних параметрів,
2. потенційне посилення атак у разі розрідженого секрету,
3. реалізаційні витоки.

3. Практична частина

3.1 Реалізація

У рамках виконання лабораторної роботи було реалізовано програмну версію решіткової криптографічної схеми SMAUG,
яка ґрунтується на складності задачі MLWR.
Реалізацію виконано мовою програмування Python з використанням стандартних бібліотек, numpy для зручної роботи з масивами, реалізовано арифметику у кільці многочленів виду $R_q = \mathbb{Z}[X]/(X^n + 1)$, генератор шуму, логіку PKE та інкапсуляції ключів (KEM) згідно зі специфікацією.

Модуль арифметики (rig)

Реалізує базову арифметику в кільці R_q: додавання/віднімання многочленів, множення (негациклічне згортання), зведення за модулем, скалярний добуток векторів та множення матриці на вектор. Також містить операції компресії/масштабування коефіцієнтів між модулями.

sampling

Огортає XOF/хеш-функції (SHAKE-128/256) та реалізує вибірку випадковостей:

- генерування матриці A з сидів через XOF;
- вибірка розріджених тернарних векторів (HWT-алгоритм зі специфікації);
- детермінований дискретний гаусівський семплер (DGS) для шуму, узгоджений із референсною реалізацією.

pke

Реалізує логіку криптосистеми з відкритим ключем SMAUG.PKE:

- KeyGen (генерація (pk=(seedA,b)) та (sk=s));
- Enc (шифрування μ з заданим сидом випадковості);

- Dec (відновлення μ).
  Логіка відповідає специфікації SMAUG з урахуванням правильного масштабу/компресії.

kem

SMAUG.KEM згідно з FO-перетворенням із специфікації:

- Encap: вибір μ → (G(\mu,H(pk))) → шифрування μ;
- Decap: розшифрування μ′, повторне шифрування, FO-перевірка, fallback-ключ.
  Серіалізація ключів і шифртекстів узгоджена з форматом специфікації; для KAT-режиму використовуються ref-сумісні пакування.

params

Описує набори параметрів для SMAUG-128, SMAUG-192, SMAUG-256, а також тестовий детермінований TOY-NOISELESS з σ=0.0 для спрощення перевірки роботи алгоритму. Також містить похідні константи (n, k, q, p, p′, t, hs, hr, σ). Забезпечує централізоване перемикання наборів параметрів.

test_smaug

Опис та запуск тестів окремих функцій, і загалом для PKE (M == Dec(Enc(M))) та декапсуляції інкапсульованого ключа.

Модуль кодеків (codec)

Реалізує пакування/розпакування коефіцієнтів у бітові строки.
Використовується для серіалізації відкритих ключів, шифртекстів і допоміжних структур у відповідності до розрядності модулів.

RNG та AES-DRBG (rng, nist_aes256ctr_drbg, nist_aes256ctr_drbg_df)

Модуль rng інкапсулює джерело випадкових байтів. Для відтворюваних тестів використовуємо AES-256 CTR-DRBG (NIST SP 800-90A) з фіксованим 48-байтовим сидом.

KAT-специфічні компоненти

- kat_runner.py: прогін офіційних KAT-векторів; підтримує режим --use-main-keygen для перевірки нашої імплементації проти референсної.
- debug_kat.py: поглиблене порівняння на одному KAT-кейсі з діагностикою проміжних значень (seed, A, s, e, b).
- У sampling і kem додані KAT-сумісні гілки (референтне пакування, ref-DGS), які використовуються лише для відтворення офіційних векторів.

3.2 Особливості референсної імплементації

Референсна реалізація (від якої отримано KAT-вектори) відрізняється від тексту статті у декількох критичних місцях. Через ці відмінності чиста реалізація за статтею не відтворює KAT-вектори. Нижче перелічено ключові розбіжності та їхній вплив.

1. KEM-перетворення (FO-потік)
    - Стаття (SMAUG.KEM, Fig. 7): Encap вибирає випадкове μ, обчислює G(μ, H(pk)), а потім шифрує μ з цим seed. Decap робить перевірку FO та fallback-ключ.
    - Референс/KAT: використовує інший потік: випадкове delta шифрується безпосередньо (indcpa_enc), а секрет формується як kdf(ct, delta).

Наслідок: навіть при повній відповідності PKE-рівня, шифртексти ct і спільні ключі ss не збігаються з KAT, якщо дотримуватись статті.

2. HWT-вибірка секрету s (можливо помилка в референсі?)
    - Стаття: кожен компонент s[i] має бути незалежним (seed змішується з i).
    - Референс: у genSx_vec nonce підготовлено, але не використано; genSx викликається з одним і тим самим seed для всіх i, тому всі s[i] однакові.
    - Наслідок: інший розподіл секрету; якщо генерувати незалежні s[i] як у статті, KAT-вектори не збігаються.
3. Формат пакування коефіцієнтів (packing)
    - Стаття: описує параметри, але не фіксує точний байтовий формат.
    - Референс/KAT: використовує конкретне пакування 10-бітних коефіцієнтів (5 байт на 4 коефіцієнти).
    - Наслідок: навіть якщо математично b однакове, pk у байтах різний.
4. Алгоритм KEM:
    - У статті: FO-перетворення через μ → G(μ,H(pk)) → Enc(pk, μ; seed).
    - У референсі/KAT: delta → indcpa_enc, ss = kdf(ct, delta)
    - Наслідок: ct/ss не збігаються з KAT при чистій реалізації за статтею.
    - Компроміс: main-реалізація лишається за статтею, а KAT-перевірка використовує референс-KEM.

Для виявлення відмінностей, окрім статичного аналізу коду референсної імплементації, ми також реалізували набір тестів окремих функцій обох виконань SMAUG і порівняння результатів, що можна побачити в модулі debug_kat.py, в якому реалізовано один KAT тест для SMAUG-128 покроково, і кожен з проміжкових результатів (A, b, sk, random, тощо) порівнюються між собою

3.3 Тести

1. PKE
    a. test_pke_roundtrip — перевірка коректності шифрування/розшифрування
    b. test_pke_wrong_key — перевірка некоректності розшифрування іншим ключем
    c. test_pke_ciphertext_tamper – перевірка незмінюваності повідомлення
    d.
2. KEM
    a. test_kem_encap_decaps — перевірка, що encaps і decaps видають однаковий ключ.
    b. test_kem_wrong_key — розшифрування «чужим» ключем дає інший shared key.
    c. test_kem_ciphertext_tamper — модифікований шифротекст змінює shared key.
    d. test_kem_replay_same_ct — повторна декапсуляція одного й того ж ct є детермінованою.
    e. test_kem_random_ct_rejection — випадковий ct не дає коректний ключ.
3. Sampler
    a. test_hwt_sampler_stats — перевіряє статистичні властивості HWT-семплера: що кожний поліном має точну вагу hs/k, а значення коефіцієнтів належать лише множині {-1,0,1}. Додатково перевіряється, що у non-KAT режимі поліноми не однакові (seed змішується з індексом).
    b. test_gaussian_sampler_stats — збирає вибірку з DGS і оцінює середнє та дисперсію
4. Codec / Compression

a. test_codec_roundtrip_and_edges — пакування та розпакування для різних бітових ширин, перевірка країв діапазону, нецілих байтів, помилкових форматів

5. Determinism
    a. test_determinism_keygen — однаковий seed → однакові pk/sk.
    b. test_determinism_encap — однаковий DRBG seed → однакові pk/ct/K.
6. Ring
    a. test_ring_add_sub_roundtrip — перевірка консистентності додавання/віднімання в Rq.
    b. test_ring_mul_identity — множення на 1 не змінює поліном.
    c. test_ring_negacyclic_wrap — перевіряє негациклічне “обгортання”: $x \cdot x^N - 1 = x^N$ у кільці R_q.
    d. test_ring_mod_reduction — перевіряє коректність зведення за модулем для значень поза діапазоном (позитивних і від’ємних).

3.4 Порівняння пропускної здатності

Оцінювання швидкодії реалізації не проводилося, оскільки метою даної роботи був аналіз алгоритмічних властивостей схеми, а не її практична ефективність. Реалізація виконувалася мовою Python, що не дозволяє отримати репрезентативні показники продуктивності. Проте з оригінальної статті SMAUG приводимо порівняння зі схожими ґратковими алгоритмами наведений в Таблиці 3.4.1

Таблиця 3.4.1

Схеми	**Тактів**			**Тактів відносно**		
	Keygen	**Encaps**	**Decaps**	**Keygen**	**Encaps**	**Decaps**
Kyber512	131 560	162 472	18 930	1,70	2,10	2,03
LightSaber	93 752	122 176	133 764	1,21	1,58	1,44
LightSable	85 274	114 822	128 990	1,10	1,48	1,39
SMAUG-128	77 220	77 370	92 916	1,00	1,00	1,00
Kyber768	214 160	251 308	285 378	1,38	1,84	1,75
Saber	187 222	224 686	239 590	1,21	1,64	1,47
Sable	170 400	211 290	237 224	1,10	1,55	1,45
SMAUG-192	154 862	136 616	163 354	1,00	1,00	1,00
Kyber1024	332 470	371 854	415 498	1,25	1,38	1,36
FireSaber	289 278	347 900	382 326	1,08	1,29	1,25
FireSable	275 156	337 322	371 486	1,03	1,25	1,22
SMAUG-256	266 704	270 123	305 452	1,00	1,00	1,00

Висновки

У ході виконання лабораторної роботи було проведено теоретичний і практичний аналіз решіткової криптографічної схеми SMAUG, що ґрунтується на складності задач MLWR/MLWE. На теоретичному рівні розглянуто математичні основи побудови, параметри та специфікацію процедур PKE і KEM, а також особливості семплінгу (HWT, dGaussian) і перетворення Фуджісакі-Окамото. На практичному етапі реалізовано модульну Python-версію SMAUG: арифметику в кільці, генерацію ґратки A, процедури семплінгу, PKE та KEM, а також механізми серіалізації. Архітектура коду є компонентною, що дозволяє окремо тестувати й замінювати ключові блоки. Коректність реалізації підтверджено системою модульних і інтеграційних тестів для арифметики, семплерів, серіалізації, детермінізму, PKE/KEM-логіки та захисту від модифікації шифртекстів. Додатково виконано узгодження з референсною реалізацією й KAT-векторами, а відмінності між специфікацією статті та референсом зафіксовано й документовано.