Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Комп'ютерної інженерії та управління
(повна назва)
Кафедра Безпеки інформаційних технологій
(повна назва)

АТЕСТАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти другий (магістерський)

Дослідження криптографічних атак на ЕЦП FALCON
(тема)

Виконав:
Черниш Д.І.
(прізвище, ініціали)
студент 2 курсу, групи БІКСм-19-1
спеціальності 125 - Кібербезпека
(код і повна назва спеціальності)
Тип програми освітньо-професійна
(освітньо-професійна або освітньо-наукова)
Освітньої програми Безпека інформаційних та комунікаційних систем
(повна назва освітньої програми)

Керівник Мельникова О.А.
(прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Халімов Г.З.
(підпис) (прізвище, ініціали)

2020 р.

Харківський національний університет радіоелектроніки

Факультет _____Комп'ютерної інженерії та управління_____
Кафедра _____Безпеки інформаційних технологій_____
Рівень вищої освіти _____магістр_____
Спеціальність _____125 Кібербезпека_____ _____
(шифр і назва)
Освітня програма_____Безпека інформаційних та комунікаційних систем_____
(повна назва)

ЗАТВЕРДЖУЮ:
Зав. кафедри БІТ Халімов Г.З.

_____
(підпис)

«_____»_____ 2020 р.

ЗАВДАННЯ
НА АТЕСТАЦІЙНУ РОБОТУ

студентові _____Чернишу Денису Ігоровичу_____
(прізвище, ім'я, по батькові)

1. Тема роботи: Дослідження криптографічних атак на ЕЦП FALCON
затверджена наказом по університету від "22" жовтня 2020 р. № 1412Ст
2. Термін подання студентом роботи 17.12.2020
3. Вихідні дані до роботи: опис алгоритму ЕЦП FALCON, опис теоретично можливих атак на криптосистеми на базі решіток NTRU
4. Зміст пояснювальної записки (перелік питань, що потрібно розробити):
1) аналіз алгоритму ЕЦП FALCON;
2) аналіз теоретично можливих атак;
3) аналіз алгоритмів приведення базису решіток;
4) дослідження можливості застосування алгоритмів приведення до FALCON;
5) програмна реалізація спроби проведення атаки з використанням алгоритмів приведення базису решітки на ЕЦП FALCON.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслеників, плакатів): презентація з графічними слайдами.

6. Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Основна частина	Доцент каф. БІТ Мельникова О.А.		

7. Дата видачі завдання 14 вересня 2020 року

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	Отримання завдання на атестаційну роботу	14.09.2020	виконано
2.	Пошук та аналіз літературних джерел за темою дипломної роботи	14.09.2020	виконано
3.	Аналіз алгоритму ЕЦП FALCON	28.09.2020	виконано
4.	Аналіз алгоритмів приведення базису решіток	14.10.2020	виконано
5.	Дослідження можливості застосування алгоритмів приведення до FALCON	01.11.2020	виконано
6.	Реалізація програми	23.11.2020	виконано
7.	Оформлення роботи	01.12.2020	виконано
8.	Представлення роботи до захисту	17.12.2020	виконано

Студент _____ Черниш Д.І.
(підпис) (прізвище, ініціали)

Керівник роботи _____ доцент Мельникова О.А.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ

Пояснювальна записка до атестаційної роботи: 87 с., 5 ч., 4 табл., 2 рис., 1 дод., 42 джерел.

ЦИФРОВИЙ ПІДПИС, ПРИВЕДЕННЯ РЕШІТОК, FALCON, ПОСТКВАНТОВА КРИПТОГРАФІЯ, ФАКТОР ЕРМІТА, РЕШІТКИ, КРИПТОАНАЛІЗ, АТАКИ

Об'єкт дослідження – атаки на алгоритм ЕЦП FALCON.

Предмет дослідження – атаки на алгоритм електронного цифрового підпису FALCON з використанням алгоритмів приведення базису решіток.

Мета роботи – дослідити можливі криптографічні атаки на алгоритм електронного цифрового підпису FALCON, дослідити сучасні алгоритми приведення базису решітки та використати їх для атаки на FALCON.

У роботі розглянуто алгоритм ЕЦП FALCON, розглянуто сучасні алгоритми приведення базису решітки, подано докладний опис та принципи роботи та реалізації самого ефективного алгоритму Self-dual BKZ.

Основні результати – проведено аналіз постквантової криптосистеми FALCON, створено програмну реалізацію алгоритму з її використанням, виконана програмна реалізація спроби атаки на FALCON з використанням різних процедур приведення базису решітки.

ABSTRACT

Explanatory note to the diploma work: 87 pages, 5 parts, 4 tables, 2 figures, 1 addition, 42 sources.

DIGITAL SIGNATURE, LATTICE REDUCTION, FALCON, POSTQUANTOUS CRYPTOGRAPHY, HERMIT FACTOR, LATTICES, CRYPTOANALYSIS, ATTACKS

The object of study – attacks on the EDS algorithm FALCON.

The subject of the research – attacks on the FALCON electronic digital signature algorithm using lattice base reduction algorithms.

The purpose of the work – to investigate possible cryptographic attacks on the FALCON electronic digital signature algorithm, to investigate modern lattice base reduction algorithms and use them to attack FALCON.

The FALCON EDS algorithm is considered in the work, modern lattice base reduction algorithms are considered, the detailed description and principles of work and realization of the most effective Self-dual BKZ algorithm are given.

The main results - the analysis of the post-quantum cryptosystem FALCON is carried out, the software implementation of the algorithm with its use is created, the software implementation of the attack attempt on FALCON with the use of various lattice base reduction procedures is performed.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ЕЦП	–	електронний цифровий підпис
NTT	–	Number Theoretic Transform, аналог FFT
ПЗ	–	програмне забезпечення
GPV	–	структура описана Джентрі, Пайкертом і Вайкунтанатаном
LLL	–	алгоритм Ленстры-Ленстры-Ловаса
SVP	–	shortest vector problem (задача знаходження найкоротшого вектору)
BKZ	–	блоковий алгоритм Коркіна-Золоторьова
DBKZ	–	подвоєний блоковий алгоритм Коркіна-Золоторьова
NTL	–	Number Theory Library
GSO	–	Ортогоналізація Грамма-Шмідта

ВСТУП

Згідно з чинним законодавством України будь-яка інформація в комп'ютерних мережах підлягає захисту: відкрита інформація повинна зберігати властивості цілісності і доступності, інформація з обмеженим доступом повинна зберігати властивості конфіденційності, цілісності та доступності.

З метою забезпечення конфіденційності даних використовують механізми симетричного і асиметричного шифрування. Для контролю цілісності можуть бути використані механізми електронного цифрового підпису.

Більшість сучасних асиметричних криптосистем побудовані на обчисленнях в кільцях, полях простих чисел і групах точок еліптичних кривих, їх стійкість базується на складності рішення задач факторизації (RSA), дискретного логарифма в простому полі (DSA) або групі точок еліптичної кривої (ECDSA).

Всі ці завдання можуть бути вирішені з використанням квантового комп'ютера достатньої обчислювальної потужності, який може бути створений в найближчому майбутньому. З метою аналізу та стандартизації нових криптосистем NIST оголошує про початок відкритого конкурсу. У першому раунді бере участь ряд алгоритмів, заснованих на різних алгебраїчних структурах - решітках, хеш-функціях і ряді інших. На конкурс було подано кілька алгоритмів ЕЦП заснованих на решітках, одним з таких є алгоритм ЕЦП FALCON [1, 39].

Існує чимала кількість криптографічних атак на сучасні криптосистеми на решітках. У тому числі в цей список криптосистем потрапляю і алгоритми електронних цифрових підписів в тому числі і алгоритм електронного цифрового підпису FALCON.

Завданням магістерської роботи є проведення дослідження теоретично можливих криптографічних атак на алгоритм електронного цифрового підпису FALCON, дослідження основних алгоритмів, що використовуються для атак на алгоритми що базуються на решітках.

Завданням практичної направленості є програмна реалізація алгоритму FALCON та спроба проведення атаки н алгоритм.

У першому розділі наведено загальні відомості щодо алгоритму ЕЦП FALCON необхідні для розуміння змісту описаних атак.

У другому розділі розглядаються теоретично можливі атаки на алгоритм ЕЦП.

У третьому розділі подано опис алгоритмів приведення базису решіток, що використовуються для проведення більшості атак.

У четвертому розділі наведено докладний опис самого ефективного алгоритму приведення базису решітки DBKZ.

У четвертому розділі подано опис програмної реалізації алгоритму FALCON та модуля, що призначений для симуляції атаки на алгоритм.

1 СТРУКТУРА АЛГОРИТМУ ЕЦП FALCON

У цьому розділі буде надана основна інформація про структуру алгоритму електронного цифрового підпису FALCON. Також буде наведено принцип роботи основних частин алгоритму таких як генерація ключів та підпису що є необхідним для розуміння принципу роботи алгоритму.

1.1 Основний принцип FALCON

Логічне обґрунтування конструкції FALCON випливає з простого спостереження: при переході з підписів на основі RSA або дискретного логарифма на пост-квантові підписи складність зв'язку, ймовірно, буде більшою проблемою, ніж швидкість. Дійсно, багато пост-квантових схем мають простий алгебраїчний опис, який робить їх швидкими, але вони завжди вимагають або ключів більшого розміру, ніж до-квантові схеми, або підписів великого розміру, або обох.

Очікується, що такі проблеми з продуктивністю будуть перешкоджати переходу від до-квантових до пост-квантових схем. Отже, основний принцип FALCON полягає в тому, щоб мінімізувати таке значення:

$$|\mathrm{pk}| + |\mathrm{sig}| = (\text{бітовий розмір відкритого ключа}) + (\text{бітовий розмір підпису})$$

Саме тому авторами використовуються решітки, які дозволяють зробити як |pk|, так і |sig| досить малими, у тому ж числі структуровані решітки. Коли мова йде про підписи на основі решітки, існує дві парадигми: Fiat-Shamir або hash-and-sign.

Обидві парадигми досягають порівнянних рівнів компактності, але hash-and-sign має цікаві властивості: структура GPV [2, 39], яка описує, як отримати

схеми підписів на hash-and-sign, безпечні в класичній і квантовій моделі оракула [2,10]. Крім того, вона володіє можливостями відновлення повідомлень [9]. Тому використовується ця структура. Як саме вона використовується, докладно описано в 2.2.

Для реалізації структури GPV використовуються решітки NTRU: як наведено в [8], вони дозволяють отримати досить компактну реалізацію структури GPV.

Крім того, вони подаються з кільцевою структурою, яка прискорює багато операцій на два порядки (наприклад, коефіцієнт $O(n/\log n)$ для перевірки підпису). Це докладно описано в пункті 2.3.

Останнім кроком був семплер з пасткою. У алгоритмі використовується семплер з пасткою, який асимптотично такий ж швидкий, як найшвидший універсальний семплер [11], і забезпечує той же рівень безпеки, що і найбезпечніший семплер [12]. Це докладно описано в пункті 2.4.

1.2 Структура GPV

У 2008 році Джентрі, Пейкерт і Вайкунтанатан [2] створили основу для отримання безпечних підписів на основі решітки. На дуже високому рівні ця структура може бути описана наступним чином:

– відкритий ключ містить матрицю повного рангу $\mathrm{A} \in \mathbb{Z}_q^{n \times m}$ (за наявності), яка породжує q-нарну решітку Λ.

– закритий ключ містить матрицю $\mathrm{B} \in \mathbb{Z}_q^{m \times m}$, яка породжує $\Lambda_q^{\perp}$, де $\Lambda_q^{\perp}$ позначає решітку ортогональну до Λ за модулем q: для будь-яких $x \in \Lambda$ і $y \in \Lambda$ маємо $\langle x, y \rangle = 0 \bmod q$. Еквівалентно, рядки A і B попарно ортогональні: $\mathrm{B} \times \mathrm{A}^t = 0$.

– для повідомлення, підпис являє собою короткий значення $\mathrm{s} \in \mathbb{Z}_q^m$, таке що $\mathrm{sA}^t = H(\mathrm{m})$, де $H : \{0,1\}^* \to \mathbb{Z}_q^n$ – геш-функція. З огляду на A,

процедура перевірки того, що s є дійсним підписом, проста: потрібно тільки перевірити, чи дійсно s короткий, і перевірити $\mathrm{s}\mathrm{A}^t = H(\mathrm{m})$.

– обчислення дійсного підпису є більш делікатною процедурою. Спочатку обчислюється довільний прообраз $\mathrm{c}_0 \in \mathbb{Z}_q^m$, який перевіряє $\mathrm{c}_0\mathrm{A}^t = \mathrm{c}$. Оскільки c_0 не обов'язково має бути коротким, а $m \geq n$, це можна зробити за допомогою стандартної лінійної алгебри. Потім B використовується для обчислення вектору $\mathrm{v} \in \Lambda_q^{\perp}$, близького до c_0. Різниця $\mathrm{s} = \mathrm{c}_0 - \mathrm{v}$ є коректним підписом: дійсно, $\mathrm{s}\mathrm{A}^t = \mathrm{c}_0\mathrm{A}^t - \mathrm{v}\mathrm{A}^t = \mathrm{c} - 0 = \mathrm{c}$, і якщо c_0 і v досить близькі, то s короткий.

У цій абстрактній формі цей опис схеми підпису не є специфічним для структури GPV: цей принцип був вперше реалізований в схемах підпису GGH [13] і NTRUSign [15]. Проте, GGH і NTRUSign страждають від атак з повним розкриттям, в той час як інфраструктура GPV, як доведено, є безпечною в класичних і квантових моделях випадкового оракула, які передбачають стійкість для деяких параметрів. Причина цього полягає в тому, що GGH/NTRUSign і платформа GPV мають радикально різні способи обчислення v в процедурі підписання.

Обчислення v в GGH і NTRUSign. У GGH і NTRUSign v обчислюється з використанням алгоритму, званого алгоритмом округлення і спочатку формалізованого Бабаєм. У цьому детермінованому алгоритмі c_0 спочатку виражається у вигляді реальної лінійної комбінації рядків B, вектор цих дійсних координат потім округляється за коефіцієнтом і знову множиться на B: в двох словах, $\mathrm{v} \leftarrow \left\lfloor \mathrm{c}_0\mathrm{B}^{-1} \right\rceil \mathrm{B}$, де $\lfloor \cdot \rceil$ позначає коефіцієнтне округлення. В кінці процедури $\mathrm{s} = \mathrm{v} - \mathrm{c}_0$ гарантовано лежить в паралелепіпеді $[-1,1]^m \times \mathrm{B}$, що дозволяє жорстко обмежити норму $\|\mathrm{s}\|$.

Проблема цього підходу полягає в тому, що кожний підпис s лежить в $[-1,1]^m \times \mathrm{B}$, і кожний s пропускає невелику інформацію про базис B. Цей факт

був успішно використаний декількома атаками [13, 6], які привели до повного злому схеми.

Обчислювання v в рамках GPV. Основний принцип GPV [2], який також є ключовою відмінністю платформи GPV і GGH / NTRUSign, полягає в способі обчислення v. Замість алгоритму округлення, структура GPV спирається на рандомізований варіант алгоритму найближчої площини [12], також сформульований Бабаєм. Як і в разі алгоритму округлення, використання алгоритму найближчій площині призвело б до витоку базису B і призвело б до повного злому схеми. Однак алгоритм Кляйна запобігає цьому: він рандомізований таким чином, що для даного m вибірка s проводиться відповідно до сферичного гаусового розподілу по здвинутій решітці $c_0 + \Lambda_q^{\perp}$. Цей метод не тільки несприйнятливий до атак, описаних вище, але також доведено, що він не дає ніякої інформації про базис B. Алгоритм Кляйна був фактично першим з сімейства алгоритмів, званих семплери з пасткою. Більш детальна інформація про семплери з пасткою подана в розділі 2.4.

1.3 NTRU решітки

Перше питання при створенні екземпляра платформи GPV – це клас використовуваних решіток. Обґрунтування дизайну, очевидно, грає велику роль в цьому. Дійсно, якщо акцент робиться на безпеку без компромісів, то логічним вибором буде використання стандартних решіток без будь-якої додаткової структури, як це було зроблено, наприклад, в схемі обміну ключами Фродо [6].

Головний принцип FALCON – компактність. З цієї причини FALCON побудований на решітках NTRU, представлених Хоффштейном, Пайфером і Сільверманом; вони поставляються з додатковою кільцевою структурою, яка не тільки дозволяє зменшити розмір відкритих ключів на коефіцієнт $O(n)$, але і прискорює багато обчислень як мінімум на коефіцієнт $O(n/\log n)$. Навіть в ширшому класі решіток над кільцями, решітки NTRU є одними з найбільш

компактних: відкритий ключ може бути приведений до одного поліному $h \in \mathbb{Z}_q[x]$ ступеня не більше $n-1$. При цьому використовується ідея Штеле і Штайнфельда [11], які показали, що структура GPV може використовуватися разом з решітками NTRU доказово безпечним способом.

Однак компактність була б марна без безпеки. З цієї точки зору решітки NTRU також мають підстави вселяти довіру, оскільки вони чинять опір екстенсивному криптоаналізу протягом приблизно двох десятиліть, і вони параметризуються таким чином, який робить їх ще більш стійкими.

1.4 Швидка вибірка Фур'є

Друге питання при створенні екземпляра платформи GPV – це семплер з пасткою. Семплер з пасткою приймає в якості вхідних даних матрицю A, пастку T, мішень c і виводить короткий вектор s, такий що $s^t A = c \bmod q$. За допомогою позначень розділу 2.2 це еквівалентно знаходженню $v \in \Lambda_q^{\perp}$, близького до c_0, тому можна байдуже посилатися під терміном «семплер з пасткою» на алгоритми, які виконують ту чи іншу задачу.

Очевидно, що для семплера важливо бути ефективним. Однак не менш важливою метрикою є «якість» зразка: чим коротше вектор s (або, що еквівалентно, чим ближче v до c_0), тим більш безпечним буде цей зразок.

1. Алгоритм Кляйна [12] приймає в якості пастки матрицю B. Він виводить вектор s норми, пропорційної $\|B\|_{GS}$, що є коротким та, отже, хорошим для безпеки.

2. Тим же чином, як алгоритм Кляйна є рандомізованою версією алгоритму найближчої площини, Пейкерт запропонував рандомізовану версію алгоритму округлення [14]. Гарна частина цього полягає в тому, що коли B має структуру над кільцями, він може працювати в часі і просторі $O(m \log m)$. Однак він виводить вектори норми, пропорційні спектральній нормі $\|B\|_2$ у B.

Це більше, ніж те, що отримується за допомогою алгоритму Кляйна, і, отже, гірше з точки зору безпеки.

3. Мічансіо і Пейкерт [15] запропонували новий підхід, в якому A і його люк побудовані таким чином, щоб забезпечити простий і ефективний відбір проб з люку. Цей підхід був узагальнений в [LW15]. Нажаль, він не має прямої сумісності з решітками NTRU, і неясно, чи можливо досягти того ж рівня компактності, що і з решітками NTRU.

4. Дукас і Перст [9] запропонували варіант алгоритму найближчій площини Бабая для решіток над кільцями. Це відбувається рекурсивним способом, який дуже схожий на швидке перетворення Фур'є, і з цієї причини вони охрестили його «швидкої найближчою площиною Фур'є». Цей алгоритм також може бути рандомізованим: він дає зразок з пасткою, який поєднує в собі якість алгоритму Кляйна, ефективність Пейкерта і може використовуватися в решітках NTRU.

З чотирьох щойно описаних підходів здається очевидним, що рандомізований варіант швидкої найближчої площини Фур'є [9] є найбільш адекватним вибором, враховуючи подане обґрунтування конструкції і решітки NTRU. З цієї причини це семплер з пасткою, який використовується в FALCON.

Таблиця 1.1 — Порівняння різних семплерів з пасткою

Семплер	Швидкий	Короткий s	Решітки NTRU
Klein [12]	Ні	Так	Так
Peikert [14]	Так	Ні	Так
Micciancio-Peikert [15]	Так	Так	Ні
Ducas-Prest [9]	Так	Так	Так

Вибір стандартного відхилення. При використанні семплера з пасткою важливим параметром, який необхідно встановити, є стандартне відхилення σ.

Якщо воно занадто низьке, то не гарантується, що семплер не розкриє секретний базис (для всіх відомих семплерів значення $\sigma = 0$ відкриває двері до навчання атакам, таким як [12, 13]). Але якщо воно занадто високе, семплер не повертає оптимально короткі вектори, і схема не настільки безпечна, як могла б бути. Таким чином, існує компроміс.

Використаний швидкий семплер Фур'є має багато спільного з семплером Кляйна, включаючи оптимальне значення σ (найкоротший, про який відомо, і не розкриває секретний базис). Згідно [16], цього достатньо для рівня безпеки та кількості запитів, встановлених NIST для прийняття $\sigma \leq 1.312\|\mathrm{B}\|_{GS}$, що у даному випадку означає $\sigma \leq 1.55\sqrt{q}$.

1.5 Основні складові FALCON як алгоритму ЕЦП

Для алгоритму електронного цифрового підпису основними та найважливішими частинами є генерація ключів, генерація підпису та перевірка підпису. З точки зору кріптоаналізу нас цікавлять процедури генерації ключів та генерація підпису. Також цікавим є структура ключей та відкриті параметри що є у алгоритму.

1.5.1 Відкриті параметри

Відкриті ключі використовують деякі відкриті параметри, які використовуються багатьма парами ключів:

а) циклотомічний многочлен $\phi \in \mathbb{Z}[x]$, який є монічним і не приводиться. У FALCON ϕ є одним з цих двох типів поліномів:

1) двійковий випадок. $\phi = x^n + 1$ при $n = 2^k$;

2) трійковий корпус. $\phi = x^n - x^{n/2} + 1$ при $n = 3 \cdot 2^k$.

б) модуль $q \in \mathbb{Z}^*$. У FALCON q може приймати будь-яке з цих двох значень (в обох випадках q вибирається так, щоб $(\phi \bmod q)$ розщеплювалося по $\mathbb{Z}_q[x]$):

1) двійковий випадок. Якщо $n = 2^k$ то $q = 12289$;

2) трійковий випадок. Якщо $n = 3 \cdot 2^k$ то $q = 18433$.

в) матеріальна оцінка $\beta > 0$.

Для ясності всі представлені параметри можуть бути опущені (наприклад, в заголовках алгоритмів), коли вони видалені з контексту.

1.5.2 Закритий ключ

Ядро закритого ключа FALCON sk складається з чотирьох поліномів $f, g, F, G \in \mathbb{Z}[x]/(\phi)$, з короткими цілочисленими коефіцієнтами, що підтверджує рівняння NTRU:

$$fG - gF = q \bmod \phi. \tag{1.1}$$

Крім того, многочлен f повинен бути оборотним в $\mathbb{Z}_q[x]/(\phi)$.

З огляду на те, що f і g такі, що існує рішення (F, G) рівняння NTRU, F і G можуть бути перераховані динамічно, але цей процес є обчислювально дорогим; тому зазвичай очікується, що щонайменше F буде зберігатися поруч з f і g (враховуючи, що f, g, F, G можуть бути ефективно перераховані).

Два додаткові елементи обчислюються з закритого ключа і можуть бути перераховані динамічно або збережені поруч з f, g, F:

– FFT-уявлення f, g, F, G, впорядковані у вигляді матриці:

$$\mathbf{B} = \begin{bmatrix} \mathrm{FFT}(g) & -\mathrm{FFT}(f) \\ \mathrm{FFT}(G) & -\mathrm{FFT}(F) \end{bmatrix}, \tag{1.2}$$

де $\mathrm{FFT}(a)$ позначає швидке перетворення Фур'є a в нижчележащому кільці (тут кільце $\mathbb{Z}[x]/(\phi)$);

– дерево FALCON T. Важливою тонкістю в FALCON є те, що природа дерева FALCON різниться в залежності від ϕ: в двійковому випадку дерево FALCON є двійковим деревом – кожен вузол має не більше двох дочірніх елементів, і в трійковому випадку дерево FALCON є потрійним деревом – кожен вузол має не більше трьох дочірніх елементів.

FFT-уявлення полінома формально складається з комплексних чисел (комплексне число зазвичай кодується як два 64-бітних значення з плаваючою точкою); проте FFT- представлення многочлена f надлишково, тому що для кожного комплексного кореня ς з ϕ його сполучений також є коренем з ϕ, і $f\left(\overline{\varsigma}\right)=\overline{f\left(\varsigma\right)}$. Отже, FFT- представлення многочлена може бути збережено у вигляді $n/2$ комплексних чисел, а B при збереженні вимагає $2n$ комплексних чисел.

1.5.3 Відкритий ключ

Відкритий ключ FALCON pk, відповідний закритому ключу $\mathrm{sk}=\left(f,g,F,G\right)$, є поліномом $h\in\mathbb{Z}_q[x]/\phi$ такий, що:

$$h=gf^{-1}\bmod\left(\phi,q\right). \tag{1.3}$$

1.5.4 Генерація ключів

Генерація пари ключів, мабуть, сама технічна частина FALCON для опису. Вона може бути хронологічно і концептуально розкладена на дві чітко розділені частини, кожна з яких містить свої власні технології, використовує різні математичні інструменти і потребує вирішення різних завдань.

Рішення рівняння NTRU. Перший етап генерації пари ключів складається з обчислення многочленів $f,g,F,G\in\mathbb{Z}[x]/\left(\phi\right)$, які перевіряють рівняння 1.3 – рівняння NTRU. Генерація f і g досить проста, але складна

частина полягає в тому, щоб ефективно обчислити многочлени F, G, щоб рівняння 1.3 було перевірено.

Щоб зробити це, було запропоновано метод, який використовує структуру вежі кілець. Використовується польова норма N, щоб відобразити рівняння NTRU на менше кільце $\mathbb{Z}[x]/(\phi')$ вежі кілець аж до $\mathbb{Z}$. Потім вирішується рівняння в $\mathbb{Z}$.

З точки зору реалізації, основна технічна складова цієї частини полягає в тому, що вона вимагає обробки многочленів з великими коефіцієнтами (кілька тисяч біт на коефіцієнт на найнижчих рівнях рекурсії).

Обчислення дерева FALCON. Як тільки згенеровані відповідні поліноми f, g, F, G, друга частина генерації ключа складається з попередньої обробки їх в адекватний формат: під адекватним мається на увазі, що цей формат повинен бути досить компактним і забезпечувати швидку генерацію підписи на ходу, FALCON дерева якраз цей адекватний формат. Щоб обчислити дерево FALCON, обчислюється **LDL*** розкладання **G**=**LDL*** матриці **G**=**BB***, де

$$\mathbf{B}=\begin{bmatrix} g & -f \\ G & -F \end{bmatrix}, \tag{1.4}$$

що еквівалентно обчисленню ортогоналізації Грама-Шмідта $\mathbf{B}=\mathbf{L}\times\mathbf{B}$. Якби використовувався семплер Кляйна в якості пристрою для відбирання проб з пасткою, знаючи $\mathbf{L}$, було б достатньо, але трохи незадовільно, так як не використовувалася б структура «вежа кілець» $\mathbb{Z}[x]/(\phi)$.

Таким чином, замість того щоб зупинятися на досягнутому, $\mathbf{L}$ зберігається в корені дерева, використовуючи оператори розщеплення, , щоб «розбити» діагональні елементи $\mathbf{D}_{ii}$ з $\mathbf{D}$ на матриці $\mathbf{G}_i$ над меншим кільцем $\mathbb{Z}[x]/(\phi')$, після чого створюються піддерева для кожної матриці $\mathbf{G}_i$ і рекурсивно починається процес розкладання і розщеплення **LDL***.

Рекурсія триває до тих пір, поки матриця **G** не отримає свої коефіцієнти в □ , які відповідають основі дерева рекурсії. Як це зробити, зазначено в розділі 3.7.3.

Основна технічна частина цього полягає в тому, що вона використовує структуру вежі кілець □ $[x]/(\phi)$, розбиваючи її елементи на менші кільця. Крім того, проміжні результати зберігаються в дереві, що вимагає точного обліку, оскільки елементи різних рівнів дерева не перебувають в одному і тому ж полі. Нарешті, з міркувань продуктивності цей крок повністю реалізований в області FFT.

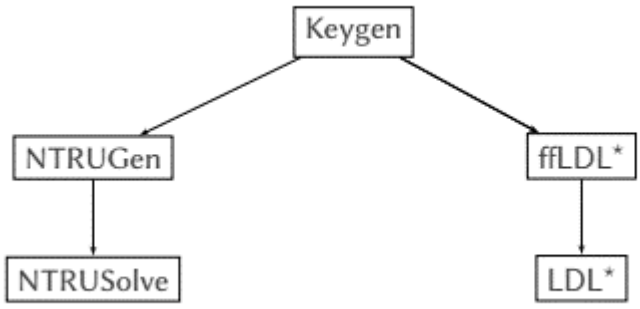

Рисунок 1.1 — Блок-схема генерації ключа

Після того, як ці два кроки виконані, решта генерації пари ключів стає простою. Останній крок нормалізує листя дерева **LDL**, щоб перетворити його в дерево FALCON. Результат обгорнутий в закритий ключ sk і відповідний відкритий ключ pk має вигляд $h = gf^{-1} \bmod q$.

Основним алгоритмом є процедура Keygen (алгоритм 1.1). Загальна архітектура генерації пари ключів також показана на рисунку 1.1.

Алгоритм 1.1Keygen(ϕ,q)

Вхідні дані: $\phi \in \mathbb{Z}[x], q$.
Вихідні дані: sk, pk.
1. $f, g, F, G, \gamma \leftarrow$ NTRUGen(ϕ,q)
2. $\mathrm{B} \leftarrow \begin{bmatrix} g & -f \\ G & -F \end{bmatrix}$
3. B ← FFT(B)
4. G ← B × B*
5. T ← ffLDL*(G)
6. if ϕ двійковий
7. $\sigma \leftarrow 1.55\sqrt{q}$
8. else if ϕ трійковий
9. $\sigma \leftarrow 1.32 \cdot 2^{1/4}\sqrt{q}$
10. для кожного leaf T
11. leaf.value ← $\sigma/\sqrt{\text{leaf.value}}$
12. sk ← $\left(\overline{\mathrm{B}}, \mathrm{T}\right)$
13. $h \leftarrow gf^{-1} \bmod q$
14. pk ← h
15. return sk, pk

1.5.5 Генерація підпису

На високому рівні принцип алгоритму генерації підпису простий: спочатку він обчислює геш-значення з $\mathbb{Z}_q[x]/(\phi)$, з повідомлення m і домішку r, а потім використовує свої знання закритого ключа f, g, F, G для обчислення двох коротких значень s_1, s_2, таких що $s_1 + s_2 h = c \bmod q$.

Наївним способом знайти такі короткі значення (s_1, s_2) було б обчислення $\mathbf{t} \leftarrow (c, 0) \cdot \mathbf{B}^{-1}$, його округлення за коефіцієнтом до вектору **z** і $(s_1, s_2) \leftarrow (\mathbf{t} - \mathbf{z})\mathbf{B}$. Можна перевірити, що (s_1, s_2) дійсно виконує всі вимоги, щоб бути легітимним, але цей метод, як відомо, небезпечний і може відбутися витік закритого ключа.

Правильний спосіб генерації (s_1, s_2) без витоку закритого ключа – використовувати семплер з пасткою. У FALCON використовується семплер з пасткою, званий швидка вибірка Фур'є.

Ядро генерації підпису, буде адаптивно застосовувати рандомізоване округлення (згідно дискретного розподілу Гауса) до коефіцієнтів **t**. Але це зроблено адаптивно, з використанням інформації, що зберігається в дереві FALCON T.

Алгоритм 1.2 ffLDL*(G)

Вхідні дані: матриця $G \in FFT\left(\left[x\right]/\left(x^n - x^{n/2} + 1\right)\right)^{n \times n}, k \in \{2,3\}$.
Вихідні дані: двійкове дерево T.

1. $(L,D) \leftarrow$ LDL*(G)
2. if $(n > 1)$
3. $\quad d_{00}, d_{01} \leftarrow$ splitfft2(D_{00})
4. $\quad d_{10}, d_{11} \leftarrow$ splitfft2(D_{11})
5. $\quad G_0 \leftarrow \begin{bmatrix} d_{00} & d_{01} \\ xd_{01} & d_{00} \end{bmatrix}, G_1 \leftarrow \begin{bmatrix} d_{10} & d_{11} \\ xd_{11} & d_{10} \end{bmatrix}$
6. $\quad$ T.value $\leftarrow L_{10}$
7. $\quad$ T.leftchild $\leftarrow$ ffLDL*(G_0)
8. $\quad$ T.rightchild $\leftarrow$ ffLDL*(G_1)
9. $\quad$ return T
10. if $(n = 6)$
11. $\quad d_{00}, d_{01}, d_{02} \leftarrow$ splitfft3(D_{00})
12. $\quad d_{10}, d_{11}, d_{12} \leftarrow$ splitfft3(D_{11})
13. $\quad G_0 \leftarrow \begin{bmatrix} d_{00} & d_{01} & d_{02} \\ xd_{02} & d_{00} & d_{01} \\ xd_{01} & xd_{02} & d_{00} \end{bmatrix}, G_1 \leftarrow \begin{bmatrix} d_{10} & d_{11} & d_{12} \\ xd_{12} & d_{10} & d_{11} \\ xd_{11} & xd_{12} & d_{10} \end{bmatrix}$
14. $\quad$ T.value $\leftarrow L_{10}$
15. $\quad$ T.leftchild $\leftarrow$ ffLDL*(G_0)
16. $\quad$ T.rightchild $\leftarrow$ ffLDL*(G_1)
17. $\quad$ return T
18. if $(n = 2)$
19. $\quad$ T.value $\leftarrow (L_{10}, L_{20}, L_{21})$
20. $\quad$ T.leftchild $\leftarrow D_{00}$
21. $\quad$ T.middlechild $\leftarrow D_{11}$
22. $\quad$ T.rightchild $\leftarrow D_{22}$
23. $\quad$ return T

На високому рівні алгоритм швидкої вибірки Фур'є можна розглядати як рекурсивний варіант семплера Кляйна (також відомого як семплер GPV). Семплер Кляйна використовує матрицю L (і норму векторів Грамма-Шмідта) в

якості пастки, в той час як використаний семплер використовує дерево таких матриць (або, скоріше, дерево їх елементів). З огляду на $\mathbf{t}=(t_0,t_1)$, алгоритм спочатку розбиває t_1, використовуючи оператор розщеплення, рекурсивно застосовує себе до нього (використовуючи правильний дочірній елемент T.rightchild з T) і використовує оператор злиття, щоб підняти рішення до бази кільця $\mathbb{Z}[x]/(\phi)$; потім він знову застосовується рекурсивно з t_0. Важливо відзначити, що рекурсії не можуть виконуватися паралельно: друга рекурсія враховує результат першої рекурсії, і це робиться з використанням інформації, що міститься в T.value.

Найбільш складною частиною алгоритму підпису є швидка вибірка Фур'є, описана в алгоритмі 3.16, тому що вона використовує дерево FALCON і дискретні гаусіани над $\mathbb{Z}$. Інша частина алгоритму, включаючи стиснення підписи, досить проста для втілювати в життя.

Формально, з огляду на закритий ключ sk і повідомлення m, підписуюча особа використовує sk для підпису m наступним чином:

– випадковий домішок **r** генерується рівномірно в $\{0,1\}^{320}$. Конкатенований рядок (**r**‖**m**) потім гешується в точку $c\in\mathbb{Z}_q[x]/(\phi)$, як визначено алгоритмом 5;

– обчислюється (не обов'язково короткий) прообраз **t** з *c*, а потім дається в якості вхідних даних для алгоритму швидкої вибірки Фур'є, який виводить два коротких полінома $s_1s_2\in\mathbb{Z}[x]/(\phi)$ (в уявленні FFT) такий, що $s_1+s_2h=q \bmod \phi$, як визначено алгоритмом 3.16;

– s_2 кодується (стискається) в ланцюжок бітів **s**;

– підпис складається з пари (**r**, **s**).

Примітка по вибірці по $\mathbb{Z}$. Алгоритм 3.16 вимагає доступу до оракула Υ для розподілу $D_{\mathbb{Z},\sigma',c'}$, де σ' може бути значенням будь-якого листа секретного дерева FALCON T, а $c'\in\mathbb{Z}$ довільне. Для реалізації потрібно

тільки, щоб розбіжність Рені між цим оракулом і ідеальним дискретним гаусовим перевіряло $\mathrm{R512}\left(\Upsilon \,||\, D_{\square,\sigma',c'}\right) \le 1+2^{-66}$ (для визначення розбіжності Рені, наведеного в [11]).

У еталонної реалізації FALCON використовується гаусовий семплер, заснований на відбракуванні з бімодальним розподілом. Слід зазначити, що діапазон можливих значень для стандартного відхилення в гаусовому семплері обмежений: він завжди більше 1,2 і завжди нижче 1,9 (у двійковому випадку) або 2,2 (в трійковому випадку).

Загальна архітектура процедури підписання показана на рисунку 1.2.

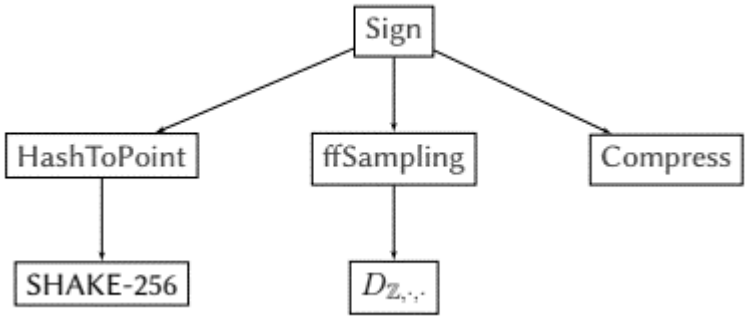

Рисунок 1.2 — Блок-схема підпису

Алгоритм 1.3 Sign(m,sk,β)

Вхідні дані: m,sk,β.
Вихідні дані: sig.

1. $\mathrm{r} \leftarrow \{0,1\}^{320}$
2. $c \leftarrow \mathrm{HashToPoint}(\mathrm{r}||\mathrm{m})$
3. $\mathrm{t} \leftarrow \left(\mathrm{FFT}(c), \mathrm{FFT}(0)\right) \cdot \mathrm{B}^{-1}$
4. do
5. $\quad \mathrm{z} \leftarrow \mathrm{ffSampling}_n(\mathrm{t},\mathrm{T})$
6. $\quad \mathrm{s} = (\mathrm{t}-\mathrm{z})\mathrm{B}$
7. while $\|\mathrm{s}\| > \beta$
8. $(s_1, s_2) \leftarrow \mathrm{invFFT}(\mathrm{s})$
9. $\mathrm{s} \leftarrow \mathrm{Compress}(s_2)$
10. return $\mathrm{sig} = (\mathrm{r},\mathrm{s})$

Знання та розуміння поданих процедур алгоритму ЕЦП FALCON дозволить більш чітко розуміти принцип атак можливих на алгоритм.

2 ВІДОМІ АТАКИ НА АЛГОРИТМ ЕЦП FALCON

В даному розділі наведено перелік і короткий опис відомих і досить ефективних атак на алгоритм електронного цифрового підпису FALCON[41].

Атаки можна поділити на два типи:

1) відновлення ключа;
2) підробка підпису.

2.1 Відновлення ключа

Існує декілька можливих атак що відносяться до цього типу. Цікавими є атаки, що базуються на алгоритмах приведення базису. До них відносяться:

1) використання алгоритму приведення решітки;
2) гібридна атака Хоугрейва-Грехема;
3) використання алгоритму BKW.

Далі буде наведено короткий опис цих атак

2.1.1 Використання алгоритму приведення решітки

Почнемо з розгляду решітки $\left(\square[x]/(\phi)\right)^2\begin{bmatrix}0 & q\\ 1 & h\end{bmatrix}$. Ця решітка є базисом підпису й містить у собі $\begin{bmatrix}f & g\end{bmatrix}$ що є секретним ключем підпису. Зміст атаки полягає в приведенні базису решітки та виконання процедури перерахування для пошуку елементів що утворюють досить короткий вектор. Атака базується на вирішенні SVP (shortest vector problem) у базисі що був приведений до розміру блоку. Проведемо розрахунки щоб розглянути можливість реалізації цієї атаки на повноцінний FALCON.

Після використання на основному базисі редукції решітки перерахуємо всі точки решітки в області радіусу $\sqrt{2n}\sigma'$ з центром на початку координат.

Таким чином, зі значною ймовірністю ми можемо знайти $[g \ f]$. Якщо ми використовуємо розмір блоку, перерахування займає незначний час, якщо норма Грамма-Шмідта більше $0.75\sqrt{B}\sigma'$. Для найвідомішого алгоритму зменшення решітки DBKZ [11], це $\left(\frac{B}{2\pi e}\right)^{(1-n/B)} \sqrt{q}$.

Тоді легко вивести B і показати, що $B = n + o(n)$. Це дає $B = 652$, при $n = 768$, і $B = 921$, при $n = 1024$. Передбачувана безпеку докладно описана в наступній таблиці з використанням методології New Hope [3].

Таблиця 2.1 — Передбачувана безпека

n	B	Класичний	Квантовий
512	392	114	103
768	652	195	172
1024	921	263	230

2.1.2 Гібридна атака Хоугрейва-Грехема

Дана атака базується на поєднанні алгоритму приведення базису решітки та атаки типу зустріч посередині (meet-in-middle). Атака ефективна проти оригінального NTRU, який використовує розріджені поліноми.

Загальновідомо, що найближчу векторну задачу (CVP) можна ефективно вирішити у випадку, якщо дана точка простору дуже близька до вектора решітки. Якщо для алгоритму вирішення CVP потрібен час *t,* а множина *S* має властивість включати принаймні одну точку $\upsilon_0 \in S$, яка дуже близька до вектора решітки, то υ_0 можна знайти за час $O\left(|S|t\right)$ перераховуючи множину *S*. Якщо точки *S* можна представити як $S = S' \oplus S'$, тобто для кожного $(\upsilon, \upsilon') \in S \times S'$ існує $\upsilon'' \in S'$ такий, що $\upsilon = \upsilon' + \upsilon''$, то існують умови, за яких насправді існує ефективний алгоритм зустрічі посередині на цьому просторі для знаходження точки υ_0 за час $O\left(|S|^{1/2} t\right)$.

Цей результат CVP можна перевести до результату щодо зменшення базису решітки, визначивши множину *S* як деякі лінійні комбінації останніх $n-m$ рядків даного базису $\{b_1\ldots,b_n\}$, а потім за допомогою алгоритму CVP на елементах *S* та базисі $\{b_1\ldots,b_m\}$. Подібний підхід застосований Шнорром [21] щодо зменшення загальних решіток за алгоритмом SHORT. Шнорр також припускає, що вдосконалення «дня народження» може бути можливим для його методу, але приходить до висновку, що загалом вимоги до зберігання можуть бути непомірними. Автори даної атаки затверджують, що використання техніки зустріч посередині може вирішити дану проблему. Атаку успішно було здійснено на NTRUEncrypt. Для FALCON даний акт є незначним через відсутність сітчастого перерахування. Також дана атака є ефективною на системи з дуже малим секретом. У FALCON розмір секрету дорівнює $\|(f,g)\|\approx\sqrt{q}$ що є дещо більшим ніж порогове значення довжини секрету для цієї атаки (дослідження показали що порогова довжина вектор дорівнює 251 при модулі коефіцієнтів поліному приблизно в 63 рази меншому ніж у FALCON).

2.1.3 Комбінаторна атака (BKW)

Комбінаторна атака BKW зазвичай використовується для LWE. Найкращим варіантом є алгоритм описаний Полом Кіршнером та П'єром Фуке[30].

При $q=O(n)$, розмір коефіцієнтів був би постійним. Тоді варіант Кирхнера-Фуке [21] BKW буде відпрацьовувати за час $2^{n/((2+o(1))\log\log n)}$, щоб відновити ключ, тобто асимптотично швидше, ніж попередні алгоритми. Це вказує на те, що найбільш компактна схема використовує $q=n^{1+\varepsilon+o(1)}$ для деякого $\varepsilon>0$. Однак, оскільки n не велике, використаного q досить, щоб зробити цю атаку неактуальною. Дійсно, навіть якщо припустити, що пошук найближчого сусіда виконується за постійний час і інші оптимістичні припущення, найкраща комбінаторна атака виконується за час 2^{135} для $n=512$.

2.2 Атаки типу підробка підпису

На даний час підробка підпису може бути здійснена шляхом знаходження точки решітки на відстані, обмеженій β від випадкової точки, в тій же решітці, що і вище. Це завдання також полегшується завдяки першому виконанню редукції решітки на вихідній основі. Одна можливість полягає в тому, щоб перерахувати всі точки решітки в кулі радіусом $\sqrt{\frac{nq}{\pi \mathrm{e}}}$. Оскільки ця куля більше, ніж в попередній атаці, ця атака буде повільніше. Може здатися, що це буде повільніше, ніж попередня атака через фактор $\Theta\left(\sqrt{n}\right)$ в радіусі. Це не той випадок, оскільки решітка має ортогональний базис, з чого випливає, що на відстані в $o\left(\sqrt{n}\right)$ мало точок $\left(2^{o(n)}\right)$. Мається на увазі, що запропонований спосіб починається з відновлення секретного ключа, так що він повільніше, ніж попередній алгоритм. Крім того, вкладення точки в решітку не допомагає: відстань до решітки $\Theta\left(\sqrt{n}\right)$ більше, ніж найкоротша ненульова точка.

2.3 Підсумок

З наведених даних можна побачити що жоден з варіантів атак неможливий на повно розмірний FALCON. Однак, атака що базується на приведенні базису решітки є однією з найімовірніших. Далі у роботі буде розглянуто сучасні алгоритми приведення базис решітки та подано опис самого ефективного на даний час алгоритму DBKZ.

3 АЛГОРИТМИ ПРИВЕДЕННЯ БАЗИСУ РЕШІТКИ

Скорочення базису решітки є фундаментальним інструментом в криптоаналізі і використовується для успішної атаки на багато криптосистеми, засновані як на решітках, так і на інших математичних задачах. Успіх даних методів в криптоаналізі в значній мірі обумовлений тим фактом, що алгоритми редукції на практиці працюють набагато краще, ніж передбачає їх теоретичний аналіз найгіршого випадку. За останні 30 років алгоритми базисного редукції досліджувалися у багатьох роботах, але розрив між теоретичним аналізом і практичної ефективністю все ще в значній мірі нез'ясовний. Цю прогалину перешкоджає здатності оцінювати безпеку криптографічних функцій на основі решітки, і він широко визнаний як один з основних перешкод на шляху використання криптографії на решітках на практиці[40].

За великим рахунком сучасний стан редукції базису решітки (в теорії і на практиці) представлено двома алгоритмами:

1) надзвичайно практичний алгоритм Шнорра і Ейхнера Block-Korkine-Zolotarev (BKZ) в його сучасному втіленні BKZ 2.0, що включає в себе стратегії обрізки, рекурсивної попередньої обробки і раннього завершення;

2) алгоритм зниження Slide Гами і Нгуєна, елегантне узагальнення LLL [6], яке доказово апроксимує короткі вектори решітки в межах факторів, пов'язаних з нерівністю Морделла.

3.1 Необхідні поняття та визначення

Позначення. Числа позначені строковими буквами що виділені курсивом. Для $n \in \mathbb{Z}_+$ множина $\{0,\ldots, n\}$ позначається як $[n]$. Для векторів використовуються строкові літери та елемент вектора v з індексом *i*

позначається через v_i. Припустимо, що $\langle v,w\rangle = \sum_i v_i \cdot w_i$ – скалярний добуток двох векторів. Якщо $p \geq 1$, p визначається нормою вектора v такою, що $\|v\|_p = \left(\sum |v_i|^p\right)^{1/p}$. В даному алгоритмі використовуються тільки норми, задані p = 1, 2 и ∞. Всякий раз, коли індекс p опускається, мається на увазі стандартна евклідова норма p = 2. Проекція вектора b, ортогонального вектору v, визначається як $\pi_v(b) = b - \frac{\langle b,v\rangle}{\|v\|^2} v$.

Матриці позначені великими літерами. Стовпець матриці B з індексом i позначається як b_i.

Окрім того, підматриця, що мстить стовпці від i-го до j-го (включно) позначається, як $B_{[i, j]}$, а горизонтальна конкатенація двох матриць B_1 и B_2 – як $[B_1|B_2]$. Для кожної матриці B та $p \geq 1$ норма визначається як $\|B\|_p = \max_{\|x\|_p=1}\left(\|Bx\|_p\right)$. Для p = 1 (або ∞) ця величина часто позначається як сума стовпців (строк); для p = 2 це також називається спектральною нормою. Класичний факт, що $\|B\|_2 \leq \sqrt{\|B\|_1 \|B\|_\infty}$. π_V(B) – це матриця, отримана шляхом застосування π_V до кожного стовпця b_i у B та $\pi_V(b_i) = \pi_{v_k}(\cdots(\pi_{v_1}(b_i))\cdots)$.

3.1.1 Решітки

Решітка Λ є дискретною підгрупою в R^m та породжується матрицею $B \in R^{m \times n}$, тобто Λ = L(B) = {Bx: x ∈ Z^n}. Якщо B має повний ранг стовпця, він називається базисом Λ, а dim (Λ) = n є розмірністю (або рангом) Λ. Решітка має нескінченно багато базисів, котрі пов’язані один з одним множенням справа з унімодулярними матрицями. З кожною матрицею B пов’язана її ортогоналізація Грамма-Шмідта (GSO) B^*, де i-й стовпець b_i^* з B^* визначається як $b_i^* = \pi_{B^*_{[1,i-1]}}(b_i) = b_i - \sum_{j<i} \mu_{i,j} b_j^*$ и $\mu_{i,j} = \langle b_i, b_j^*\rangle / \|b_j^*\|$ µi, (и $b_1^* = b_1$). Для кожного базису нескінченно багато базисів, що мають однакову кількість GSO b_1^*, серед яких є базис, котрий мінімізує $\|b_1\|$ для усіх i. Перетворення базису у цю форму

зазвичай відомо як зменшення базису та його легко й ефективно можна зробити використовуючи невелику модифікацію процесу Грамма-Шмідта.

Для фіксованої матриці B продовжується процедура проекції до індексів: $\pi_i(\cdot) = \pi_{\mathrm{B}^*_{[1,i-1]}}(\cdot)$, тому $\pi_1(\mathrm{B}) = \mathrm{B}$. Всякий раз, коли виконується посилання на форму базису B, мається на увазі вектор $\left(\left\|\mathrm{b}_i^*\right\|\right)_{i\in[n]}$. $\mathrm{D}^\dagger$ визначається як GSO для D у зворотному порядку.

Для кожної решітки Λ існує декілька інваріантів, пов'язаних з нею. Одним з них є її визначник $\det\left(L(\mathrm{B})\right) = \prod_i \left\|\mathrm{b}_i^*\right\|$ для будь-якого базису B. Незважаючи на те, що базис решітки не визначений однозначно, детермінант існує й він ефективно розраховується для базису. Окрім того, для кожної решітки Λ довжина її найкоротшого ненульового вектору (також відомого як перший мінімум) позначається через $\lambda_1(\Lambda)$, котрий завжди правильно визначений. Використовуються скорочені позначення $\det(\mathrm{B}) = \det(L(\mathrm{B}))$ та $\lambda_1(\mathrm{B}) = \lambda_1(L(\mathrm{B}))$. Теорема Мінковського є класичним результатом, котрий пов'язує перший мінімум з визначником решітки. У ньому говориться, що $\lambda_1(\Lambda) \le \sqrt{\gamma_n}\det(\Lambda)^{1/n}$ для кожного Λ с $\dim(\Lambda) = n$, де $\Omega(n) \le \gamma_n \le n$ – постійна Ерміта. Знайти найкоротший ненульовий вектор в решітці *P*-важко при рандомізованих редукціях [7].

Для кожної решітки Λ її подвійний елемент визначається як $\hat{\Lambda} = \left\{\mathrm{w} \in \mathrm{span}(\Lambda) \mid \langle \mathrm{w},\mathrm{v}\rangle \in \mathrm{Z} \text{ для всех } \mathrm{v} \in \Lambda\right\}$. Класичним фактом є те, що $\det(\Lambda) = \det(\Lambda)\hat{}\,-1$. Для базису решітки B, нехай D буде єдиною матрицею, що задовольнює $\mathrm{span}(\mathrm{B}) = \mathrm{span}(\mathrm{D})$ та $\mathrm{B}^{\mathrm{T}}\mathrm{D} = \mathrm{D}^{\mathrm{T}}\mathrm{B} = \mathrm{I}$. Тоді $L(\mathrm{B}) = L(\mathrm{D})$ та D позначається як подвійний базис B. З цього витікає, що для кожного вектору $\mathrm{w} = \mathrm{Dx}$ існує такий $\mathrm{B}^{\mathrm{T}}\mathrm{w} = \mathrm{x}$, тобто можна відновити коефіцієнти x з w використовуючи D шляхом множення з транспонуванням первинного базису B^{T}. При поданому базисі решітки її подвійний базис розраховється за поліноміальний час, але потребує не менше $\Omega(n^3)$ бітових операцій з

використанням інверсії матриці. Якщо D – подвійний базис B, їх GSO пов’язані вдношенням $\left\|\mathrm{b}_i^*\right\| = 1/\left\|\mathrm{d}_i^\dagger\right\|$.

3.1.2 Алгоритми перерахування

Для вирішення SVP на практиці зазвичай використовуються алгоритми перерахування, оскільки вони є найбільш ефективними алгоритмами для реальних на даний час вимірів. Стандартна процедура перерахування може бути описана як рекурсивний алгоритм: в якості вхідних даних використовується базис $\mathrm{B} \in \mathrm{Z}^{\mathrm{m}\times\mathrm{n}}$ та радіус *r*, спочатку вона рекурсивно знаходить усі вектори $\mathrm{v}' \in L\left(\pi_2\left(\mathrm{B}\right)\right)$ з $\left\|\mathrm{v}'\right\| \le r$, а далі для кожного з них знаходить усі $\mathrm{v} \in L\left(\mathrm{B}\right)$, де $\pi_2\left(\mathrm{v}\right) = \mathrm{v}'$ и $\left\|\mathrm{v}\right\| \le r$, використовуючи b_1.

Це, загалом, є пошуком у ширину по великому дереву, де шарами є базисні вектори, а вузли – відповідні коефіцієнти. У той час як концептуально проще уявити перерахування як BFS, реалізації зазвичай використовують глибинний пошук у цілях підвищення продуктивності.

Існує декілька практичних вдосконалень цього алгоритму, відомих як перерахування SchnorrEuchner. По-перше, завдяки симетрії решіток можливо зменшити простір пошуку, гарантуючи, що останній ненульовий коефіцієнт завжди позитивний. Крім того, якщо знайти вектор коротше кордону *r*, можна оновити останній. Тепер можна перерахувати коефіцієнти базисного вектора в порядку довжини результуючого (спроектованого) вектору i, таким чином, збільшити ймовірність раннього знаходження деякого короткого вектора, що оновить кордон *r* і зменшить простір пошуку.

3.1.3 Приведення решітки

На відміну від точних алгоритмів SVP, приведення решітки апроксимує найкоротший вектор. Якість їх виходу зазвичай вимірюється по довжині найкоротшого вектору, котрий вони можуть знайти по відношенню до кореневого визначника решітки. Ця величина позначається коефіцієнтом Ерміта – $\overline{\delta} = \left\|\mathrm{b}_1\right\| / \det\left(\mathrm{B}\right)^{1/n}$. Коефіцієнт Ерміта залежить від розміру решітки *n*,

але досліди показують, що кореневий фактор Ерміта $\delta = \overline{\delta}^{1/n}$ сходиться до константи при збільшенні *n* для розповсюджених алгоритмів редукції.

Алгоритм LLL [6] є алгоритмом поліноміального скорочення часу. Базис $B \in Z^{m \times n}$ може бути визначений як скорочений LLL, якщо B [1,2] зменшений SVP та π_2(B) зменшений LLL. Виходячи з цього, можливо довести, що при зменшенні LLL досягається кореневий актор Ерміта не більший $\delta \le \gamma_2^{1/4} \approx 1.0746$. Однак, як повідомляється, LLL поводить себе набагато краще на практиці.

BKZ є узагальненням LLL на більший розмір блоку. Базис B зменшується на BKZ с розміром блоку *k* (позначається BKZ-*k*), якщо $B_{[1,\min(k,n)]}$ скорочується SVP, а π_2(B) скорочється на BKZ-*k*. BKZ досягає цього, просто скануючи базис зліва направо та SVP, зменшуючи кожен блок *k* (або менше, коли він досягає кінця) що проектується, використовуючи оракул SVP для усіх вимірів ≤ *k*. Він повторює цей процес (який зазвичай називається обходом) до тих пір, поки не відбудеться ніяких змін. Коли *k* = *n*, це зазвичай називають зменшенням HKZ та еквівалентно рішенню SVP. Наступна оцінка для фактора Ерміта вірна для b_1 скороченого базису BKZ-*k*:

$$\|b_1\| \le 2\gamma_k^{\frac{n-1}{2(k-1)}+\frac{3}{2}} \det(B)^{1/n} \qquad (3.1)$$

Рівняння 3.1 показує, що кореневий фактор Ерміта, досягнутий за допомогою BKZ-*k*, не більший $\le \gamma_k^{\frac{1}{2(k-1)}}$. Окрім цього, хоча й немає поліноміальної прив’язки до числа викликів, котрі BKZ виконує до оракулу SVP, Ханрот, Пухоль та Штелє показали, що можна завершити BKZ після поліноміальної кількості викликів оракула SVP й до сих пір доказуємо досягнути (2.1). Нарешті, багаторазово повідомлялося, що він поводить себ дуже добре на практиці. За цими причинами BKZ дже часто використовується

на практиці, та реалізації легко доступні у різних бібліотеках, наприклад. в NTL або fpLLL [8].

Гама й Нгуєн запропонували інший алгоритм спрощення, а саме зменшення Slide. Він також параметризований розміром блока k, котрий потребується для ділення розміру решітки, він використовує оракул SVP тільки у вимірі k. Базис B визначається як зменшений Slide, якщо $B_{[1,\ k]}$ зменшений SVP, $\pi_2(B_{[2,\ k+1]})$ є подвійним SVP (якщо $k>n$) та $\pi_{k+1}(B_{[k+1,n]})$ зменшується Slide. Зменшення Slide, як описано в [9], зменшує базис, спочатку почергово SVP спрощуючи усі блоки $\pi_{ik+1}(B_{[ik+1,(i+1)k]})$ й запускаючи LLL для B. Як тільки більше не відбуваються зміни, блоки $\pi_{ik+2}(B_{[ik+2,(i+1)k+1]})$ є подвійними скороченими SVP. Увесь цей процес повторюється до тих пір, поки не відбдеться ніяких змін у результаті ітерації. По закінченню, базис гарантовано задовольнить

$$\|b_1\| \le \gamma_k^{\frac{n-1}{2(k-1)}} \det(B)^{1/n} \tag{3.2}$$

Це трохи краще, ніж рівняння (2.1), але гарантований коефіцієнт Ерміта також гарантовано буде менший за $\gamma_k^{\frac{1}{2(k-1)}}$. Спрощення Slide має бажані властивості, що полягають в тому, щоб робити тільки поліноміальну кількість викликів оракула SVP й щоб усі виклики були у вимірі k (а не в більш низьких вимірах). Останнє дозволяє проводити більш чіткий аналіз, наприклад, у поєднанні з гаусовою евристикою. На жаль, у дослідах повідомлялось, що зменшення Slide значною мірою поступається BKZ, тому воно рідше використовується на практиці, й невідомо про будь-як доступну реалізацію.

3.1.4 Гаусова евристика

Гаусова евристика дає приближення числа елементів решітки у «гарній» підмножині R^n. Більш конкретно, вона говорить, що для даної множини S й решітки Λ мається $|S\cap\Lambda|\approx \mathrm{vol}(S)/\det(\Lambda)$.

Евристика виявилась дуже корисною у середньому випадку аналізу алгоритмів побудованих на решtках. Наприклад, її можна використовувати для оцінки складності алгоритмів перерахування або у якості виходу алгоритмів спрощення решітки [8]. Алгоритми редукції працюють шляхом багаторазового обчислення найкоротшого вектору деякій решітці й вставки цього вектору у певну позицію базису. Далі починається гаусова евристика: використовуючи наведену вище формулу, можна оцінити, наскільки великим повинен бути радіус *n*-мірного шара. Використовуючи формулу об'єму для *n*-мірного шара, можна отримати оцінку для найкоротшого ненульового вектору в решітці Λ:

$$GH(\Lambda) = \frac{\left(\Gamma\left(n/2+1\right)\cdot \det(\Lambda)\right)^{1/n}}{\sqrt{\pi}} \qquad (3.3)$$

*Я*кщо k є цілим числом, $GH(k)$ визначається як гасова евристика (рівняння 2.3) для багатовимірних решіток з одиничним визначником. Евристика була перевірена на дослідах також у контексті зменшення решітки було виявлено, що вона надто груба у невеликих вимірах, але достатньо чітка починаючи з розміру > 45. Фактично, для точного визначення випадкових решіток можна показати, що очікуване значення першого мінімуму решітки (за вибором решітки) сходиться до рівняння (2.3) при прагненні розміру решітки до нескінченності.

Евристика. Для даної решітки Λ, $\lambda_1(\Lambda) = GH(\Lambda)$.

3.2 Порівняння алгоритмів редукції Slide та BKZ

Обидва алгоритми використовують оракул найкоротшої векторної задачі (SVP) для решіток меншого розміру й параметризованих межею k (що називається «розміром блока») на розмірності цих решіток. Алгоритм скорочення Slide має багато привабливих особливостей: він робить тільки

поліноміальний кількість викликів оракула SVP, всі виклики SVP відносяться до спроектовані підрешітках в точно тому ж вимірі k й він досягає найкращої відомої верхньої межі найгіршого випадку для довжини його найкоротшого результуючого вектору: $\gamma_k^{(n-1)/(2(k-1))}\det(L)^{1/n}$, де $\gamma_k = \Theta(k)$ – постійна Ерміта, а $\det(L)$ – визначник решітки. На жаль, повідомлялось, що у дослдних експериментах алгоритм зменшення Slide кращий за BKZ, який видає набагато більш короткі вектори для того ж самого розмір блока. Фактично, зазначається, що навіть BKZ з розміром блока $k = 20$ дає кращі зменшені базиси, ніх зменшення Slide з розміром блока $k = 50$. Як висновок, алгоритм зменшення Slide ніколи не використовується на практиці й його не було реалізовано та перевірено на практиці.

С іншого боку, хоча алгоритм BKZ на випадок практичний у дослідних оцінках, він також має свої недоліки. У первинному вигляді для BKZ навіть не відомо, що він завершиться після поліноміального числа звернень до оракулу SVP, й повідомляється, що його час виконання поліноміально зростає у вимрі решітки, навіть коли розмір блока встановлено на деяке відносно мале значення $k \approx 30$. Навіть після завершення найкращі доказові кордони якості виходу BKZ гірше, ніж для зменшення Slide, що найменш, на поліноміальний коефіцієнт [6]. На практиці, для вирішення проблеми часу виконання, BKZ досить часто використовується зі стратегією «дострокового завершення» [8], котра намагається евристично виявити, коли більше не очікується прогресу у виконанні алгоритм.

Теоретичні оцінки якості виведення після поліноміального числа ітерацій були доведені, але вони гірше, ніж зменшення Slide ще більшими поліноміальними факторами. Іншим фактом, який ускладнює аналіз (як в теорії, так і на практиці) якості виведення BKZ, є той факт, що алгоритм робить виклики SVP у всіх вимірах аж до розміру блоку. Теоретично це призводить до формули, яка Залежить від всіх найгірших (ермітових) констант γ_i при $i \leq k$. На практиці якість та час роботи оцінюються симулятором [8], котрий спочатку

намагається численно оцінити продуктивність оракула SVP на випадкових решітках в усіх можливих вимірах до *k*.

Одним з основних висновків є те, що алгоритм зменшення Slide є більш практичним, ніж спочатку передбачалося, й за мірою збільшення розміру він працює майже як BKZ, й у той же час пропонує просту замкнуту формулу для оцінки якості виходу. Це забезпечує простий й ефективний метод оцінки впливу атак зі зменшенням базису решітки на криптографію що базується на решітках без необхідності запуска симуляторів або інших комп'ютерних програм [8]. Ключом до даних висновків є нова процедура перерахування найкоротших векторів решітки у подвійних решітках без необхідності явного розрахунку подвійного базису. Цікавим фактом є те, що процедура подвійного перерахування майже ідентична (синтаксично) до стандартної процедури перерахування для пошуку коротких векторів у (первинній) решітці, й вона така ж ефективна на практиці. Використовуючи цю процедуру, можна проводити дослідження, використовуючи зменшення Slide зі значно більшим розміром блока, ніж повідомлялося раніше, й спостерігається, що різниця між теоретичними (більш передбачуваними) алгоритмами й практичною евристикою збільшується.

Цей алгоритм запропонований Даніелом Міччіансіо та Майклом Вальтером у 2016 році та набув назву DBKZ (або Self-Dual BKZ). Цей алгоритм на даний момент є найкращим серед усіх відомих алгоритмів приведення базису. Тому пропонується розглянути цей алгоритм більш детально.

Докладний опис цього алгоритму та експериментальні результати його дослідження будуть наведені у наступному розділі.

4 АЛГОРИТМ ПРИВЕДЕННЯ БАЗИСУ РЕШІТКИ DBKZ

Найефективнішим на даний час алгоритмом приведення базису решітки є алгоритм DBKZ (Self-Dual BKZ). Даний алгоритм поєднує у собі переваги класичного BKZ та реалізує процедуру дострокового завершення пошуку. Тобто зменшує кількість звернень до оракула тим самим зменшуючи складність та час роботи.

У цьому розділі будуть наведені характеристики алгоритму, принцип його роботи а також результати досліджень його на практиці та теоретична оцінка очікуваних результатів для більших «розмірів блока».

4.1 Основні відомості

Для блоків невеликого розміру (скажімо, до 40) все ще існує значний розрив між якістю на виході зменшення Slide і BKZ на практиці. Для цих параметрів в [11] представлений новий варіант BKZ, заснований на подвійності решітки і новому понятті базисної редукції. Новий алгоритм DBKZ може бути ефективно реалізований з використанням процедури подвійного перерахування, досягаючи часу виконання, який можна порівняти з BKZ, і майже точно зіставляючи його експериментальне якість виведення для невеликого розміру блоку. У той же час алгоритм має різні переваги перед BKZ, що робить його найкращою метою для теоретичного аналізу: він робить виклики тільки до оракула SVP в фіксованому вимірі k, і він сам подвійний в тому, що він виконує, по суті, ті ж самі операції. Той факт, що всі виклики SVP на проектованих підрешітках знаходяться в одній фіксованій розмірності k, має декілька важливих наслідків. По-перше, це призводить до більш простої оцінці довжини найкоротшого вихідного вектору, яка може бути виражена як функція

тільки γ_k. Що ще більш важливо, це дозволяє отримати практичну оцінку якості продукції, просто замінивши γ_k значенням, передбаченим гаусовою евристикою $GH(k)$, що зазвичай використовуються у криптоанализі алгоритмів що базуються на решітках. Зазначається, що формула $GH(k)$ була перевірена для відносно великих значень k, де вона дає достатньо чіткі оцінки найменшої довжини вектору у k-мірних підрешітках.

Однак ранні роботи по прогнозуванню редукції решітки також показали, що при малих k (скажімо, до $k \leq 25$) підрешітки BKZ не дотримуються гаусової евристики. В результаті, хоча симулятор BKZ 2.0 з [8] широко використовується $GH(k)$ для великих значень k, він також повинен вдатися до громіздких експериментальних оцінок для прогнозування результату викликів SVP в вимірі нижче k. Виконуючи лише SVP-виклики на k-мірних підрешітках, алгоритм усуває необхідність в будь-яких таких експериментальних оцінках і дозволяє прогнозувати вихідна якість (при тих же або більш слабких евристичних припущеннях, ніж симулятор BKZ 2.0), просто використовуючи $GH(k)$. Підкреслимо, що це вірно не тільки для довжини найкоротшого вектору, знайденого цим алгоритмом, але можна оцінити ще багато властивостей отриманого базису. Це важливо в багатьох криптоаналітичних установках, де скорочення решітки використовується в якості попередньої обробки для інших атак. Зокрема, використовуючи гаусову евристику, ми можемо показати, що велика частина базисного виведення за алгоритмом може слідувати припущенню геометричного ряду, часто не скупиться припущення про вихід зменшення решітки, але до сих пір ніколи не доведено. Останнє потенційне перевагу, яка полягає в тому, щоб робити виклики SVP тільки з фіксованим виміром k (і, отже, можливістю використовувати гаусову евристику для всіх з них), полягає в тому, що це відкриває можливість ще більш точного стохастичного моделювання (або аналітичні рішення) , де детермінована формула $GH(k)$ замінюється розподілом ймовірностей (по довжині найкоротшого вектору в випадкової k-мірній решітці).

Алгоритми перерахування (які зазвичай використовуються при скороченні базису) знаходять короткі вектори в решітці, досліджуючи всі можливі координати $x_1,...,x_n$ можливих коротких векторів решітки $\sum_i \mathrm{b}_i \cdot x_i$ щодо заданого базису решітки та використання довжини спроектованого вектору решітки для скорочення пошуку. Представлений алгоритм перерахування подвійний решітки працює аналогічно, але без явного обчислення основи для подвійний решітки. Ключова технічна ідея полягає в тому, що можна перерахувати по скалярним творам $y_i = \langle \mathrm{b}_i, \mathrm{v} \rangle$ можливих коротких подвійних векторів v і векторів первинного базису b_i. Можливо, що дивно, можна також обчислити довжину проекцій вектора подвійний решітки v (необхідного для скорочення дерева перерахування) без явного обчислення v або подвійного базису. Простота алгоритму найкраще ілюструється, якщо просто подивитися на псевдокод і порівняти його з боку в бік з псевдокодом стандартного (первинного) перерахування решіток. Дві програми майже ідентичні, що призводить до подвійного перерахуванню, яке настільки ж ефективно, як первинне перерахування, і дозволяє застосовувати всі стандартні оптимізації (наприклад, всі різні форми обрізки), які були розроблені для перерахування в первинних решітках.

На основі скорочення базисів алгоритм DBKZ заснований на новому понятті базисно-скорочених базисів. Як і для BKZ, DBKZ-скорочення найкраще описати як рекурсивне визначення. Фактично, рекурсивне умова по суті однаково для обох алгоритмів: враховуючи базис B, якщо b є найкоротшим вектором в підрешітці, породженим першими *k* базисними векторами $\mathrm{B}_{[1,\ k]}$, потребується проекція B, ортогональна до b, щоб задовольнити рекурсивне властивість скорочення. Різниця між BKZ і DBKZ полягає в тому, що, хоча BKZ вимагає, щоб $\mathrm{B}_{[1,\ k]}$ починався з найкоротшого вектору решітки $\mathrm{b} = \mathrm{b}_1$, в DBKZ потрібно, щоб він закінчувався найкоротшим двоїстим вектором.

Цей простий поворот у визначенні приведеного базису призводить до набагато більш простому обмеження довжини b, покращуючи найбільш відому оцінку скорочення BKZ і відповідаючи теоретичного якості зменшення Slide.

Як і BKZ, Self-Dual BKZ параметризрвані розміром блоку k і оракулом SVP в вимірі k і діє на вхідний основі $B \in Z^{m \times n}$, повторюючи обходи. Початок кожного туру точно так же, як тур BKZ, тобто SVP скорочує кожен блок $\pi_i(B_{[i,\, i+k-1]})$ з $i = 1$ до $n - k + 1$. Назвемо цю частину туром вперед. В останньому блоці, який BKZ просто скорочує HKZ і звідки випливає більшість проблем для значущих передбачень, ми робимо щось інше. Замість цього ми подвоюємо SVP останній блок і продовжуємо подвійним SVP, зменшуючи все блоки розміру k назад (що є зворотним обходом). Після ітерації цього процесу (який ми називаємо туром по Self-Dual BKZ) алгоритм завершується, коли прогрес більше не досягається. Алгоритм формально описаний в Алгоритмі 4.1.

Алгоритм 4.1 Self-Dual BKZ

DBKZ(B, k, SVP_k)

Вхід: базис $B \in Z^{m \times n}$, розмір блока k, SVP оракул зменшений по k

Вихід: зменшений по k базис B'

1 do

2 for $i = 1 ... n - k$

3 SVP зменшення $\pi_k\left(B_{[i,i+k-1]}\right)$ з використанням SVP_k

4 for $i = n - k + 1 ... 1$

5 подвійне SVP зменшення $\pi_k\left(B_{[i,i+k-1]}\right)$ з використанням SVP_k

6 while є прогрес

7 return B

Зверніть увагу, що, як і BKZ, Self-Dual BKZ (DBKZ) є належним узагальненням блоку алгоритму LLL, що відповідає випадку $k = 2$. Умова завершення в рядку 6 спеціально залишено неоднозначним в цій точці, оскільки існує декілька розумні способи підійти до цього. Потрібно бути обережним, з

одного боку, гарантувати припинення, в той час як з іншого боку досягати остаточного визначення стислості.

4.1.1 Аналіз

Вихідні дані алгоритму 4.1 задовольняють наступного визначення редукції після завершення:

Визначення 4.1. базис B = [b_1, ..., b_n] *k*- скорочений, якщо або n <k, або він задовольняє таким умовам: $\left\|b_k^*\right\|^{-1} \le \lambda\left(L\left(B_{[1,k]}\right)\right)$. Для деякого SVP-скороченного базиса $\overline{B}$ з $L\left(B_{[1,k]}\right), \pi_2\left(\left[\overline{B} \mid B_{[k+1,n]}\right]\right)$ *k*-скорочений.

Для початку доводиться, що алгоритм 4.1 дійсно досягає визначення 4.1, коли використовується з конкретним умовою завершення:

Лемма 4.1. Нехай B - n-мірний базис. якщо π_{k+1}(B) однаково до і після одного циклу алгоритму 2.1, то B зменшується на k.

Доказ Лемми 4.1. Доказ індуктивний: при $n = k$ результат тривіально вірний. Передбачається, що $n > k$, і що результат вже виконується для n - 1. В кінці кожної ітерації перший блок $B_{[1,\ k]}$ є подвійним SVP, скороченим з побудови. Таким чином, потрібно тільки перевірити, що для деякого $\overline{B}$, SVP приведений базис для $L(B_{[1,\ k]})$, $\pi_2\left(\left[\overline{B} \mid B_{[k+1,n]}\right]\right)$ також *k*-приведений. Нехай $\overline{B}$ – приведений базис SVP, отриманий на першому етапі. Перша і остання операція в циклі не змінюють $L(B_{[1,\ k]})$ та $B_{[k+1,\ n]}$. Звідси слідує що π_{k+1}(B) Однаково до і після часткового туру (обхід без першого і останнього кроку) на проектованої основі $\pi_2\left(\left[\overline{B} \mid B_{[k+1,n]}\right]\right)$, й тому π_{k+2}(B) однаково до і після часткового туру. За припущенням індукції $\pi_2\left(\left[\overline{B} \mid B_{[k+1,n]}\right]\right)$ *k*-скорочено.

Лемма 4.1 дає умова завершення, яке гарантуватиме зменшення базису. Так само можна адаптувати доказ так, щоб було достатньо перевірити, що форма проектованого базису π_{k+1}(B) однакова до і після туру, що набагато ближче до того, що можна було б перевіряти, чи був досягнутий прогрес. Однак

це вимагає деякого ослаблення визначення SVP-редукції, так що перший вектор не обов'язково є найкоротшим вектором, а просто коротким вектором, що досягає межі Міньковського. Щоб показати, що вихідна якість Self-Dual BKZ в гіршому випадку, щонайменше, так само добре, як і поведінку BKZ в гіршому випадку, проводиться аналіз фактора Ерміта, якого він досягає:

Теорема 4.1. Якщо B зменшено k, то $\lambda_1\left(\mathrm{B}_{[1,k]}\right) \le \sqrt{\gamma_k}^{\frac{n-1}{k-1}} \cdot \det\left(\mathrm{B}\right)^{1/n}$. Доказ теореми 2.1. Передбачається, що L(B) має визначник Δ_1, та нехай Δ визначник $L(\mathrm{B}_{[1,\,k]})$. Нехай $\lambda \le \sqrt{\gamma_k}\Delta^{1/k}$ и $\hat{\lambda} \le \sqrt{\gamma_k}\Delta^{-1/k}$ – довжини найкоротших ненульових простих і подвійних векторів в $L(\mathrm{B}_{[1,\,k]})$. Далі доводиться, що $\lambda \le \sqrt{\gamma_k}^{\frac{n-1}{k-1}}$.

За індукцією по n спочатку показується, що визначник Δ_1 перших $k-1$ векторів не більший за $\det\left(\mathrm{B}\right)^{(k-1)/n} < \sqrt{\gamma_k}^{n-k+1}$. Оскільки B є k-скороченим, цей визначник дорівнює $\Delta_1 = \hat{\lambda} \cdot \Delta \le \sqrt{\gamma_k}\Delta^{1-1/k}$. (Це вже доказує базовий випадок індукції для $n = k$.) Тепер, нехай $\overline{\mathrm{B}}$ – скорочений SVP-базис $L(\mathrm{B}_{[1,\,k]})$, задовольняючий визначенню k-скорочення, розглядається визначник $\Delta_2 = \Delta / \lambda$ від $\pi_2\left(\overline{\mathrm{B}}\right)$. Оскільки $\pi_2\left(\left[\overline{\mathrm{B}} \mid \mathrm{B}_{[k+1,n]}\right]\right)$ має визначник $1/\left\|\overline{\mathrm{b}}_1\right\| = 1/\lambda$, за припущенням індукції є $\Delta_2 \le \sqrt{\gamma_k}^{n-k}\left(1/\lambda\right)^{(k-1)/(n-1)}$.

$$\Delta = \lambda\Delta_2 \le \sqrt{\gamma_k}^{n-k}\lambda^{\frac{n-k}{n-1}} \le \sqrt{\gamma_k}^{n-k}\left(\sqrt{\gamma_k}\Delta^{1/k}\right)^{\frac{n-k}{n-1}} = \sqrt{\gamma_k}^{\frac{(n-k)n}{n-1}}\Delta^{\frac{n-k}{k(n-1)}}$$

Піднімаючи обидві сторони до ступеня $(n-1)/n$, отримуємо $\Delta^{1-1/n} \le \sqrt{\gamma_n}^{n-k}\Delta^{\frac{1}{k}-\frac{1}{n}}$ що, що дорівнює, $\Delta^{1-1/k} \le \sqrt{\gamma_k}^{n-k}$. Звідси витікає, що $\Delta_1 = \hat{\lambda}\Delta \le \sqrt{\gamma_k}\Delta^{1-1/k} \le \sqrt{\gamma_n}^{n-k+1}$, завершуючи доказ по індукції. Тепер є можливим довести основне твердження теореми. З індуктивного доказу $\Delta \le \sqrt{\gamma_k}^{n-k}\lambda^{\frac{n-k}{n-1}}$. отже, $\lambda \le \sqrt{\gamma_k}\Delta^{1/k} \le \sqrt{\gamma_k}^{n/k}\lambda^{\frac{n-k}{k(n-1)}}$. рішення для λ, доказує теорему 4.1.

4.2 Динамічна система

Домогтися гарного часу роботи для DBKZ безпосередньо так само складно, як і на BKZ, тому в цьому розділі аналізується алгоритм DBKZ, використовуючи метод динамічної системи з [10].

Нехай $\mathrm{B}=[\mathrm{b}_1,...,\mathrm{b}_n]$ – вхідний базис для DBKZ, det(B)=1. Під час прямого проходу алгоритм розраховує послідовність векторів решітки $\mathrm{B}'=[\mathrm{b}'_1,...,\mathrm{b}'_{i-1}]$, де кожному b'_i заданий найкоротший вектор в проекції $\left[b_i,...,b_{i+k-1}\right]$, ортогональній до $[\mathrm{b}'_1,...,\mathrm{b}'_{i-1}]$. Цей набір векторів можна розширити до базису для початкової решітки. Так як $[\mathrm{b}'_1,...,\mathrm{b}'_{i-1}]$ генерує примітивну підрешітку з $\left[b_i,...,b_{i+k-1}\right]$, у згенерованої підрешітці є визначник $\det\left(L\left(b_i,...,b_{i+k-1}\right)\right)/\det\left(L\left(\mathrm{b}'_1,...,\mathrm{b}'_{i-1}\right)\right)$, а довжина його найкоротшого вектору дорівнює

$$\left\|\left(\mathrm{b}'_i\right)^*\right\| \le \sqrt{\gamma k}\left(\frac{\det\left(L\left(b_i,...,b_{i+k-1}\right)\right)}{\det\left(L\left(\mathrm{b}'_1,...,\mathrm{b}'_{i-1}\right)\right)}\right)^{1/k} \tag{4.1}$$

На цьому етапі моделювання, засноване на гаусовій евристиці, зазвичай передбачає, що (4.1) виконується з рівністю. Щоб отримати строгий аналіз без евристичних припущень, використовується метод амортизації з [10]. Для кожного $i=1,...,n-k$, нехай $x_i=\log\det\left(b_1,...,b_{1+k-1}\right)$ та $x'_i=\log\det\left(b'_1,...,b'_i\right)$. Використовуючи (4.1), отримаємо для усіх $i=1,...,n-k$,

$$\begin{aligned} x'_i &= x'_{i-1}+\log\left\|\left(\mathrm{b}'_1\right)^*\right\| \\ &\le x'_{i-1}+\alpha+\frac{x'_i-x'_{i-1}}{k}, \\ &= \omega x'_{i-1}+\alpha\left(1-\omega\right)x_i \end{aligned}$$

де $\omega = (1-1/k), \alpha = \frac{1}{2}\log\gamma_k$ и $x'_0 = 0$. За індукцією на *i*,

$$x'_1 \le \alpha\frac{1-\omega^i}{1-\omega} + (1-\omega)\sum_{j=1}^{i}\omega^{i-j}x_j$$

або у матричному записі $x^i \le$ b+Ax , де

$$\text{b}=\alpha\text{k}\begin{bmatrix}1-\omega\\ \cdots\\ 1-\omega^{n-k}\end{bmatrix}\quad \text{A}=\frac{1}{k}\begin{bmatrix}1 & & & \\ \omega & 1 & & \\ \cdots & \cdots & \cdots & \\ \omega^{n-k-1} & \cdots & \omega & 1\end{bmatrix}$$

Оскільки усі елементи A позитивні, можна побачити, що якщо $\text{X}_i \ge \text{x}_i$ є верхніми оцінками початкових значень x_i для усіх *i*, то вектор X^i=AX+b дає верхні оцінки вихідних значень $x'_i \le X'_i$.

Вектор x' описує форму базисної матриці перед виконанням зворотнього проходу. Використовуючи решітчасту подвійність, зворотній маршрут може бути еквівалентно сформульований наступними кроками:

1. Розрахувати наступний зворотний базис D з B`.
2. Застосувати прямий прохід до D, щоб отримати новий подвійний базис D`.
3. Розрахувати новий подвійний базис D`.

Звернене подвійне базисне обчислення дає базис D такий, що для всіх *i*=1,...,*n*−*k*,

$$\begin{aligned} y_i &= \log\det(d_1,...,d_{k+i-1}) \\ &= -\log\left(\det(\text{B}')/\det\left(\left[b'_1,...,b'_{n-k+1-i}\right]\right)\right) \\ &= \log\det\left(\left[b'_1,...,b'_{n-k+1-i}\right]\right) = x'_{n-k+1-i}\end{aligned}$$

Вектор y, що описує форму подвійного базису на початку зворотного ходу, є зворотною стороною x_i. Звідси випливає, що застосування повного (прямого і зворотного) обходу DBKZ створює основу, так що якщо X є верхніми межами лог-визначників x вхідний матриці, то лог-визначники вихідний матриці обмежуються зверху

R(AR(AX+b)+b)=(RA)^2X+(RA+I)Rb,

де R – матриця перестановок координат. Це призводить до вивчення дискретного часу однієї динамічної системи

$$X \mapsto \left(\mathrm{RA}\right)^2 X + \left(\mathrm{RA} + \mathrm{I}\right)\mathrm{Rb} \tag{4.2}$$

4.2.1 Якість вихідних значень

Спочатку необхідно довести, що ця система має не більше однієї фіксованої точки.

Ствердження 4.1. Динамічна система (4.2) має не більше одної фіксованої точки.

Доказ ствердження 4.1. Будь-яка фіксована точка є рішенням лінійної системи ((RA)^2 − I)X+(RA+I)Rb=0. Щоб довести єдиність, показується, що матриця ((RA)^2−I) не є єдиним числом. Якщо (RA)^{2}x=x, то x = 0. Матриця RA симетрична, тому (RA)^2=(RA)$^{\mathrm{T}}$RA=A$^{\mathrm{T}}$A. Таким чином, доказ того, що ((RA)^2−I) не є єдиним, еквівалентно тому, щоб показати, що дане значення не є власним значенням A$^{\mathrm{T}}$A. Мається $\rho\left(\mathrm{A}^{\mathrm{T}}\mathrm{A}\right) = \left\|\mathrm{A}\right\|_2^2 \le \left\|\mathrm{A}\right\|_1 \left\|\mathrm{A}\right\|_\infty$ ρ, де $\rho(\cdot)$ позначає спектральний радіус даної матриці (тобто найбільше власне значення в абсолютній величині). Але також є

$$\left\|\mathrm{A}\right\|_\infty = \left\|\mathrm{A}\right\|_1 = \frac{1}{k}\sum_{i=0}^{n-k-1}\omega^i = \frac{1-\omega^{n-k}}{k\left(1-\omega\right)} = 1-\omega^{n-k} < 1 \tag{4.3}$$

з котрого видно, що абсолютне значення будь-якого власного значення A$^{\mathrm{T}}$A строго менше за 1.

Далі необхідно знайти фіксовану точку для (4.2). (RA)^2−I – не одинична матриця. Так як (RA)^2 −I=(RA+I)(RA−I), звідси маємо, що (RA±I) також не одиничні. Таким чином, можна викреслити (RA+I) з рівняння з фіксованою точкою ((RA)^2 −I)x+(RA+I)Rb=0 та отримати (RA−I)x+Rb=0. Це показує, що

єдина фіксована точка повної динамічної системи (якщо вона існує) також повинна бути фіксованою точкою прямого ходу $x \mapsto R(Ax+b)$.

Ствердження 4.2. Нерухома точка динамічної системи $x \mapsto R(Ax+b)$ визначається як

$$x_i = \frac{(n-k-i+1)(k+i-1)}{k-1}\alpha \tag{4.4}$$

Доказ ствердження 4.2. Єдина фіксована точка системи задається рішенням лінійної системи (R−A) x=b. Далі доказується, що (4.4) є рішенням системи за індукцією за строками. Для першої строки система дає

$$x_{n-k} - x_1 / k = \alpha \tag{4.5}$$

З (4.5) виходить, що $x_{n-k} = \frac{n-1}{k-1}\alpha$ и $x_1 = \frac{k(n-k)}{k-1}\alpha$. Підставляючи їх до (4.5), рівняння легко перевіряється.

r-й ряд системи задається

$$x_{n-k-r+1} - \frac{1}{k}\left(\sum_{j=1}^{r} \omega^{r-j} x_j\right) = \frac{1-\omega^r}{1-\omega}\alpha \tag{4.6}$$

що еквівалентно

$$x_{n-k-r+1} + \omega\left(x_{n-k-r+2} - \frac{1}{k}\left(\sum_{j=1}^{r-1} \omega^{r-j-1} x_j\right)\right) - \frac{x_r}{k} - \omega x_{n-k-r+2} = \frac{1-\omega^r}{1-\omega}\alpha \tag{4.7}$$

За припущенням індукції це еквівалентно

$$\omega\left(\frac{1-\omega^{r-1}}{1-\omega}\right)\alpha + x_{n-k-r+1} - \frac{x^r}{k} - \omega x_{n-k-r+2} = \frac{1-\omega^r}{1-\omega}\alpha \qquad (4.8)$$

Підставляючи (4.5) в (4.8) для $i=n-k-r+1$, r, та $n-k-r+2$, отримаємо

$$x_{n-k-r+1} - \frac{x_r}{k} - \omega x_{n-k-r+2} =$$
$$= \frac{kr(n-r) - (n-r-k+1)(r+k-1) - (k-1)(r-1)(n-r+1)}{k(k-1)}\alpha$$

Це, у свою чергу, показує, що ліва частина (4.8) еквівалентна

$$\omega\left(\frac{1-\omega^{r-1}}{1-\omega}\right)\alpha + \alpha$$

котра дорівнює його правій частині.

Оскільки x_1 відповідає log-визначнику першого блока, застосування теореми Мінковського призводить до того ж факту Ерміта найгіршого випадку, який доведений в теоремі 4.1.

4.2.2 Конвергенція

Будь-який вхідний вектор v можна записати як v=x+e, де x – це фіксована точка динамічної системи, як й у (2.7). Система відправляє v до $v \mapsto RAv+b=RAx+RAe+b=x+RAe$, тому різність e у фіксованій точці відображається в RAe на кожній ітерації. Для аналізу збіжності алгоритму розглядається індуктована норма матриці $\|RA\|_p = \|A\|_p$, оскільки після t ітерацій розбіжність дорівнює $(RA)^t e$, тому його норма обмежена $\left\|(RA)^t e\right\|_p = \leq \left\|(RA)^t\right\|_p \|e\|_p \leq \|RA\|_p^t \|e\|_p$. Таким чином, якщо індуктована норма A строго менша за 1, відповідна норма вектору помилок слід за експоненціальним загасанням. Зокрема, в (2.6) можна побачити, що $\|A\|_\infty = 1-\omega^{n-k}$. Це доводить, що алгоритм збігається. Окрім того, нехай входом є базис B (с det(B)=1), відповідний вектор $v=\left(\log\det\left(b_1,...,b_{k+i-1}\right)\right)_{1\leq i\leq n}$ й записується v=x+e. Тоді маємо $\|e\|_\infty = \|v-x\|_\infty \leq \|v\|_\infty + \|x\|_\infty \leq \mathrm{poly}\left(n, \mathrm{size}(B)\right)$. Це позначає, що для

$$\mathrm{t} = \mathrm{polylog}\left(n, \mathrm{size}(\mathrm{B})\right) / \omega^{n-k} \approx O\left(e^{(n-k)/k}\right) \mathrm{polylog}\left(n, \mathrm{size}(\mathrm{B})\right) \qquad (4.9)$$

маємо $\left\|(\mathrm{RA})^t \mathrm{e}\right\| \le c$ для постійної c. Рівняння 4.9 вже показує, що для $k=\Omega(\mathrm{n})$ алгоритм збігається у ряді полілогаріфмічних циклів у розмірності n решітки, тобто виконує не більше $\tilde{O}(n)$ викликів SVP. У первинному варіанті роботи [11] доказ збіжності поліномів для довільного k було залишено відкритим.

Ствердження 4.3. Нехай x – фіксована точка динамічної системи, як зазначено у ствердженні 2.2 та $r_i = e_i / x_i$ – відносна похибка динамічної 4системи. Тоді, якщо r, r` – відносна похибка до та після виконання одної ітерації системи, то $\|\mathrm{r}'\|_\infty \le (1-\varepsilon)\|\mathrm{r}\|_\infty$ для $\varepsilon = 1/\left(1+n^2/\left(4k(k-1)\right)\right) \approx (2k/n)^2$ Коли $k \ge n/2$, достатньо узяти $\varepsilon = (k-1)/(n-1)$.

Доказ ствердження 4.3. Нехай $\|\mathrm{r}\|_\infty = (k-1)/\alpha$, т. е. $|e_i| \le \left((k-1)/\alpha\right) x_i = (n-k-i+1)(k+i-1)$ для усіх $i=1,...,n-k$. Далі доказується, що $|r'_j| = \left|(\mathrm{RAe})_j\right| / x_j \le \left((k-1)/\alpha\right)(1-\varepsilon)$ для усіх j або $\left|(\mathrm{Ae})_j\right| = \left|(\mathrm{RAe})_{n-k-j+1}\right| \le \frac{k-1}{\alpha}(1-\varepsilon) x_{n-k-j+1} = j(n-j)(1-\varepsilon)$.

За визначенням A $\left|(\mathrm{Ae})_j\right| \le \frac{1}{k}\sum_{i=1}^{j} \omega^{j-i}(n-k-i+1)(k+i-1) \equiv f(j)$

Достатньо довести, що функція $f(j)$ у правій частині задовільнює $f(j) \le j(n-j)(1-\varepsilon)$. Далі доказується, що ця нерівність за індукцією по j у припущенні, що $\varepsilon \le g(j) \equiv \frac{k(k-1)}{k(n-2j-1)+j(n-j)}$,

1. базовий випадок (j=1): $f(1) = n-k \le (n-1)(1-\varepsilon)$. тоді й тільки тоді, коли $\varepsilon \le g(0) = (k-1)/(n-1)$,

2. індуктивний крок: припускається, що $f(j) \le j(n-j)(1-\varepsilon)$ за індуктивною гіпотезою. Необхідно довести, що

$$f(j+1) = \omega f(j) + \frac{(n-k-j)(k+j)}{k} \le \frac{k-1}{k} j(n-j)(1-\varepsilon) + \frac{(n-k-j)(k+j)}{k}$$

не більше $(j+1)(n-(j+1))(1-\varepsilon)$. Це вірно, тільки якщо $\varepsilon \le g(j)$.

Це завершує індуктивний доказ, якщо $\varepsilon \le g(j)$для j=0,...,($n-k$). Нарешті, можна побачити що функція $g(j)$ мінімізується при $j = n/2-k$, з мінімумом $g\left(\frac{n}{2}-k\right) = 1/\left(1+n^2/\left(4k(k-1)\right)\right)$. Коли $k > n/2$, мінімум досягається при $j < 0$, а $g(j)$ монотонно зростає при $j \le 0$. Тому достатньо узяти $\varepsilon = g(0) = (k-1)/(n-1)$.

За аналогічним аргументом, описаним вище, це показує, що похибка може бути зроблена довільно близькою до 0 в$O\left((n/2k)^2\right)\text{polylog}\left(n,\text{size}(\text{B})\right)$ проходах.

4.3 Еврістичний аналіз

В контексті криптоанализа найбільший інтерес представляє поведінка алгоритмів в середньому випадку. Для цього може бути використано дуже просте спостереження, щоб передбачити фактор Ерміта, досягнутий DBKZ. Доказ теореми 4.1 заснований виключно на оцінці Мінковського $\lambda_1(\text{B}) \le \sqrt{\gamma_n}\det(\text{B})^{1/n}$. Заміна його евристичної одиницею призводить до наступного слідству.

Наслідок 4.1. Застосування евристики до кожної решітці, яка передається оракула SVP під час виконання алгоритму 4.1, якщо B скорочено k, то $\lambda_1(\text{B}_{1,k}) = GH(k)^{\frac{n-1}{k-1}}\det(\text{B})^{1/n}$.

Оскільки фактор Ерміта є найбільш значущою величиною в багатьох криптоаналітичних середовищах, наслідку 4.1 вже досить для багатьох передбачуваних застосувань з точки зору якості. Доказ досягнення найгіршої

якості виведення при зменшенні Slide також спирається тільки на оцінку Міньковського. Це означає, що те ж саме спостереження можна використовувати для прогнозування середнього поведінки випадку зменшення Slide і дає ту ж оцінку, що і наслідок 4.1. Фактично, з рекурсивного визначення скорочення Slide ясно, що це дає ще більше інформації про повернутий базисі: ми можемо використовувати наслідок 4.1 для передбачення норми $\|\mathrm{b}_{ik+1}\|$ для усіх $i \in [n/k]$. Короткий розрахунок показує, що ці вектори слідують за геометричним рядом, підтверджуючи часто передбачуване поведінку редукції решітки, а саме припущення про геометричному ряді.

Однак багато атаки вимагають набагато більш точної оцінки середнього результату. На щастя, застосування аналогічного трюку, як у наслідку 4.1, до аналізу динамічних систем дозволяє отримати набагато більше інформації про базисі. Для цього варто відзначити, що можна замінити теорему Маньківського в аналізі на евристику. Це перетворення лише незначно змінює динамічну систему в (4.2), єдина відмінність полягає в тому, що $\alpha = \frac{1}{2}\log GH(k)$. Оскільки аналіз не Залежить від постійної α, можна перевести фіксовану точку в (4.4) в інформацію про форму базису, яку DBKZ, ймовірно, поверне.

Наслідок 4.2 Застосування евристики до кожної решітці, переданої оракула SVP під час виконання алгоритму 4.1, фіксованої точки евристичної динамічної системи, тобто (4.3) с $\alpha = \frac{1}{2}\log GH(k)$, дорівнює (4.4) з тим же самим α й має на увазі, що після ще одного обходу базис задовольняє

$$\|\mathrm{b}_i^*\| = GH(k)^{\frac{n+1-2i}{2(k-1)}} \det(L(\mathrm{B}))^{1/n} \tag{4.10}$$

для усіх $i \le n-k$.

Доказ наслідку 4.2. Згідно (4.5), після завершення алгоритму 4.1 вихідний базис задовольняє $\log\left(\det\left(\left[\mathrm{b}_1,...,\mathrm{b}_i\right]\right)\right)=\frac{(n-k-i+1)(k+i-1)}{k-1}\alpha$.

За еврістикою маємо $\log\left\|\mathrm{b}_1\right\|=\alpha+x_1/k$, з котрого легко отримати рівняння при i = 1. Тепер припустимо, що (4.10) виконується для усіх $j < i$. Тоді, знову ж за еврістикою, $\log\left\|\mathrm{b}_i^*\right\|=\alpha+\left(x_i-\sum_{j<i}\log\left\|\mathrm{b}_j^*\right\|/k\right)$. Викликаючи гіпотезу індукції, рівняння (4.10) легко виходить для усіх $i \le n-k$.

Наслідок 4.2 показує, що вихідні данні алгоритму DBKZ, якщо він завершується після прямого проходу, будуть точно слідкувати по GSA, щонайменше, для усіх $i \le n-k$ й можуть бути розраховані з використанням простих замкнутих формул. Слід зазначити, що самодвоїсті властивості DBKZ мають на увазі, що в разі припинення після зворотного обходу GSA виконується для всіх $i > k$. Це позначає, що у Залежно від програми можна вибрати, яку частину вихідний бази прогнозувати. DBKZ дозволяє прогнозувати набагато більшу частину базису, ніж зменшення Slide, виключно на основі гаусовой евристики. Очевидно, що ті ж самі припущення можуть використовуватися для оцінки решти частин форми основи в разі зменшення Slide і DBKZ, оскільки остаточне застосування зменшення HKZ до окремих блоках розміру k вимагає лише незначної кількості часу в порівнянні з часом роботи всього алгоритму. Крім того, оскільки оцінка відомої частини форми (з наслідку 4.1 та 4.2) НЕ Залежить від цих додаткових припущень, оцінка зменшення Slide і DBKZ набагато менш чутлива до правильності цих припущень, в той час як помилки поширюються під час моделювання BKZ.

4.4 Подвійне перерахування

Подібно до зменшення слайдів, DBKZ інтенсивно використовує подвійне зменшення SVP проектованих блоків. Очевидним способом

досягнення цього зменшення є обчислення подвійної бази для прогнозованого блоку,

запустіть на ній основне зменшення SVP і, нарешті, обчисліть основну основу блоку. Хоча перехід між первинною та подвійною основою є поліноміальним обчисленням часу, і, таким чином, домінує етап перерахування, він передбачає інверсію матриці, що на практиці може зайняти досить багато часу. Для вирішення цього питання Гама та Нгуєн [15] запропонували іншу стратегію. Зверніть увагу, що зменшення SVP, виконане шляхом перерахування, складається з двох етапів: обчислюються координати найкоротшого вектора в даному базисі, і цей вектор вставляється в основу. Гама та Нгуєн відзначають, що для подвійного зменшення SVPможна досягти, використовуючи координати, отримані під час подвійного перерахування, працюючи виключно на первинній основі. Крім того, зауважте, що процедура переліку діє лише на GSO бази, тому її достатньо для того щоб інвертувати матриці GSO спроектованого блоку, що, навпаки, простіше, оскільки вони складаються з діагоналі та верхньої трикутної матриці. Однак це все ще вимагає обчислювальних витрат $\Omega(n^3)$.

Зараз ми вводимо спосіб знайти координати найкоротшого вектору в подвійній решітці без обчислення подвійного базису або подвійного GSO. Як розминку зверніть увагу, що процедура перерахування, застосована до подвійного базису, обчислює всі можливі коефіцієнти коротких векторів у подвійному базисі. Ми можемо перерахувати ці коефіцієнти $x_i = \langle \mathrm{v}, \mathrm{b}_i \rangle \in \square$ найкоротшого подвійного вектору v безпосередньо і обмежити складність цього підходу наступним чином: $x_i = \langle \mathrm{v}, \mathrm{b}_i \rangle = \langle \mathrm{v}, \mathrm{b}_i^* \rangle - \sum_{j<i} \mu_{i,j} \langle \mathrm{v}, \mathrm{b}_j^* \rangle$

Позначимо $x_j^* = \langle \mathrm{v}, \mathrm{b}_j^* \rangle$, котрий може бути розрахований з $(x_1,\ldots,x_j)$, отримаємо $x_i + \sum_{j<i} \mu_{i,j} x_j^* = \langle \mathrm{v}, \mathrm{b}_i^* \rangle$ та далі $\left| x_i + \sum_{j<i} \mu_{i,j} x_j^* \right| \le \|\mathrm{v}\| \|\mathrm{b}_i^*\| \le \frac{\lambda_1}{\|\mathrm{d}_i^\dagger\|}$ що саме те, що можна було б очікувати, тобто воно відповідає складності перерахування

всіх подвійних точок решітки в паралелепіпеді. Це просте спостереження припускає, що існує алгоритм для обчислення координат найкоротшого подвійного вектору, настільки ж ефективного, як і для найкоротшого первинного вектору. Однак без подвійного базису не існує очевидного способу оцінити якість розчину, тобто довжини подвійного вектору з урахуванням лише його координат у подвійному базисі. Крім того, не маючи можливості оцінити частковий розв'язок, ми можемо перераховувати лише паралелепіпеди, тоді як класичне перерахування в первинному просторі здатне перераховувати точки решітки в кулі, віднімаючи довжину часткових розчинів з радіуса перерахування. Наступна лема вирішує цю проблему і призведе до алгоритму подвійного перерахування, який сильно нагадує первинну процедуру перерахування.

Лемма 4.2. Нехай B – базис решітки, а w – довільний вектор у лінійному проміжку B. Нехай x – вектор коефіцієнта, що виражає w відносно подвійного базису, тобто $x_i = \langle \mathrm{v}, \mathrm{b}_i \rangle$ для усіх $i \le n$. Тоді для будь-якого $k \le n$ вектор $\mathrm{w}^{(k)} \in \mathrm{span}\left(\mathrm{B}_{[1,k]}\right)$ такий що $\langle \mathrm{w}^{(k)}, \mathrm{b}_i \rangle = x_i$ для усіх $i \le k$, може бути представлений як $\mathrm{w}^{(k)} = \sum_{i \le k} a_i \mathrm{b}_i^* / \left\| \mathrm{b}_i^* \right\|^2$ де

$$a_i = x_i - \sum_{j<i} \mu_{i,j} a_i \,. \tag{4.11}$$

Доказ Лемма 4.2. Умова $\mathrm{w}^{(k)} \in \mathrm{span}\left(\mathrm{B}_{[1,k]}\right)$ безпосередньо випливає з визначення $\mathrm{w}^{(k)} = \sum_{i \le k} a_i \mathrm{b}_i^* / \left\| \mathrm{b}_i^* \right\|^2$. Потрібно показати, що цей вектор також задовольняє умовам скалярного добутку $\langle \mathrm{w}^{(k)}, \mathrm{b}_i \rangle = x_i$ для всіх $i \le k$. Підставивши вираз для $\mathrm{w}^{(k)}$ у скалярний добуток, отримаємо

$\left\langle \mathrm{w}^{(k)}, \mathrm{b}_i \right\rangle = \sum_{j \le k} a_j \frac{\left\langle \mathrm{b}_j^*, \mathrm{b}_i \right\rangle}{\left\| \mathrm{b}_j^* \right\|^2} = \sum_{j \le i} a_j \frac{\left\langle \mathrm{b}_j^*, \mathrm{b}_i \right\rangle}{\left\| \mathrm{b}_j^* \right\|^2} = a_i + \sum_{j<i} a_j \mu_{i,j} = x_i$ де остання рівність випливає з визначення a_i.

Це показує, що якщо перерахувати рівні від *k*=1 до *n*, можна легко обчислити a_k з усіх заданих або раніше обчислених величин в *O(n)*. Довжина $\mathrm{w}^{(k)}$ задана як

$$\left\| \mathrm{w}^{(k)} \right\|^2 = \sum_{i \le k} a_i^2 / \left\| \mathrm{b}_i^* \right\|^2 = \left\| \mathrm{w}^{(k-1)} \right\|^2 + a_k^2 / \left\| \mathrm{b}_k^* \right\|^2 . \qquad (4.12)$$

Щоб отримати алгоритм, який є практично таким же ефективним, як первинне перерахування, необхідно застосувати ті самі стандартні оптимізації, відомі як перерахування SchnorrEuchner, до подвійного перерахування. Очевидно, що ми можемо використовувати симетричність решітки та оновлення динамічного радіуса таким же чином, як і в первинному перерахуванні. Єдиною оптимізацією, яка є не зовсім очевидною, є перерахування значень x_k в порядку збільшення довжини результуючого часткового рішення. Однак з рівняння (4.11) та (4.12) ясно, що ми можемо почати з вибору $x_k = \left\lceil \sum_{j<k} \mu_{k,j} a_j \right\rfloor$, щоб мінімізувати перше значення α_k, а потім продовжувати, рухаючись навколо цього першого значення як і в алгоритмі первинного перерахування SchnorrEuchner.

Алгоритм 4.2 Dual Enumeration

DualEnum($\mu, \left(\left\|b_i^*\right\|^2\right)_{i\in[n]}, A$)

Вхід: GSO $\mu, \left(\left\|b_i^*\right\|^2\right)_{i\in[n]}$, верхня межа A

Вихід: координати найкоротшего вектора у подвійному базисі D

1 $k \leftarrow 1$

2 while $k \geq 1$

3 $a_k \leftarrow x_k - \sum_{j<k} \mu_{k,j} a_j$

4 $l_k \leftarrow l_{k-1} + a_k^2 / \left\|b_k^*\right\|^2$

5 if $l_k \leq A$ and $k = n$ then

6 s $\leftarrow$ x, $A \leftarrow l_k$

7 if $l_k \leq A$ and $k < n$ then

8 $k \leftarrow k+1, x_k \leftarrow \left\lceil \sum_{j<k} \mu_{k,j} a_j \right\rceil$

9 else

10 $k \leftarrow k-1, x_k \leftarrow \text{nextX}(k)$

11 return s

Також примітно, що можливість обчислення часткових рішень дозволяє навіть застосовувати обрізання [14] безпосередньо. Підсумовуючи це, це показує, що подвійне перелічення SVP повинно бути таким же ефективним, як і основне перерахування. Щоб проілюструвати це, алгоритм 4.2 і 4.3 показують варіант SchnorrEuchner для процедури подвійного перерахування.

Алгоритм 4.3 Primal Enumeration

PrimalEnum($\mu, \left(\left\|\mathrm{b}_i^*\right\|^2\right)_{i\in[n]}, A$)

Вхід: GSO $\mu, \left(\left\|\mathrm{b}_i^*\right\|^2\right)_{i\in[n]}$, верхня межа A

Вихід: координати найкоротшего вектора у базисі B

1 $k \leftarrow 1$

2 while $k \geq 1$

3 $a_k \leftarrow x_k + \sum_{j>k} \mu_{k,j} a_j$

4 $l_k \leftarrow l_{k+1} + a_k^2 \left\|\mathrm{b}_k^*\right\|^2$

5 if $l_k \leq A$ and $k = 1$ then

6 $\mathrm{s} \leftarrow \mathrm{x}, A \leftarrow l_k$

7 if $l_k \leq A$ and $k > 1$ then

8 $k \leftarrow k-1, x_k \leftarrow \left\lceil -\sum_{j<k} \mu_{k,j} a_j \right\rceil$

9 else

10 $k \leftarrow k+1, x_k \leftarrow \mathrm{nextX}(k)$

11 return s

4.4.1 Особливості реалізації

Щоб навести кілька експериментальних доказів того, що подвійне перерахування настільки ж ефективне, як і основне перерахування, воно було застосовано при використанні бібліотеки NTL. Алгоритм 2 можна легко додати до реалізації алгоритму 4.3, використовуючи особливі випадки при доступі до даних та кілька операцій. Крім того, щоб уникнути поділу в рядку 4, можна попередньо обчислити необхідне значення для всіх k. Реалізація була порівнена з первинним переліченням на 10 випадкових базах у розмірності $35 \leq n \leq 50$. Як і очікувалося, швидкість перерахування була близькою до рівної в обох випадках: близько 3,2 – 107 вузлів в секунду. Невеликі розбіжності можна пояснити змінною кількістю перелічених вузлів, і таким чином певні витрати

на встановлення амортизуються за різною кількістю вузлів. Отримані результати можна побачити у таблиці 4.1.

Таблиця 4.1 — Результати роботи алгоритмів 4.2 та 4.3

N	35	40	45	50
Primal	2.73	3.16	3.13	3.17
Dual	2.77	3.19	3.18	3.27

З таблиці 4.1 можна побачити що різниця у часі для двох алгоритмів не надто велика, незважаючи на те, що алгоритм 4.2 робить вдвічі більше проходів у найгіршому випадку для знаходження найкоротшого вектору.

Даний алгоритм у поєднанні з Self-dual BKZ був використаний для перевірки можливості реалізації атаки типу відновлення ключа на алгоритм FALCON(Розділ 5).

5 ПРОГРАМНА РЕАЛІЗАЦІЯ АТАКИ НА FALCON З ВИКОРИСТАННЯМ АЛГОРИТМІВ ПРИВЕДЕННЯ БАЗИСУ РЕШІТКИ

У даному розділі наведений опис програмної реалізації алгоритму FALCON та наведені результати роботи та опис програми що симулює атаку на алгоритм електронного цифрового підпису FALCON. Також будуть подані порівняння алгоритму DBKZ з найвідомішими алгоритмами що представленні у бібліотеці NTL: LLL та BKZ. Дослідження проводилися на FALCON різних розмірностей але усі розмірності є ступенем двійки щоб не порушувати основну структуру алгоритму.

5.1 Загальні відомості

В ході дослідження алгоритму FALCON та атак на нього на базі алгоритмів приведення базису решітки, було розроблено два програмних модулі “FALCON” та “Test attacks on FALCON”.

Програмний модуль “FALCON” надає користувачу інтерфейс для роботи з алгоритмом. Він дозволяє користувачу згенерувати нову пару ключів, ініціалізувати FALCON з існуючими ключами, записати ключі до файлової системи, підписати файл, записати підпис у файл, перевірити файл з ініціалізованим FALCON та перевірити підпис використовуючи файл відкритого ключа. Модуль “Test attacks on FALCON” дозволяє провести процедуру приведення базису решітки алгоритмами LLL, BKZ та Self-dual BKZ використовуючи параметри отримані з алгоритму FALCON та використовуючи довільні параметри генерації решіток.

Під час розробки програм була використана бібліотека з відкритим кодом – NTL[37] та fplll[38]. У цих бібліотеках реалізовано можливість працювати з алгебраїчними решітками та реалізовані алгоритми BKZ та LLL. Бібліотека NTL використана як одна з частин програмного модулю “Test attacks

on FALCON”. Бібліотека fplll була використана для генерації тестових решіток та перевірки коректності реалізації алгоритму Self-dual BKZ. Так як бібліотека fplll не надає можливості прямого підключення її до проекту а використовується як окрема програма, було прийнято рішення про використання бібліотеки NTL як основної бібліотеки лдля роботи з решітками та алгоритмами приведення базису решітки.

Усі компоненти написані на мові програмування C/C++ з використанням засобу бібліотеки NTL у середовищі розробки Microsoft Visual Studio 2019. Для їх запуску необхідно мати операційну систему Windows. Для компіляції MS Visual Studio 2014 та вище. Програма тестувалася на ПК з ОС Windows 10, MS Visual Studio 2019, процесор Intel® Core i5 CPU 7500 @3.40GHz, 8 ГБ ОЗП.

5.2 Функціональне призначення та обмеження

Програмний модуль “FALCON” призначений для:

- генерації ключів FALCON;
- генерації підпису;
- перевірки підпису.

Програмний модуль “Test attacks on FALCON” призначений для:

- застосування алгоритмів приведення базису решітки для параметрів відкритого ключа алгоритму;
- застосування алгоритмів приведення базису решітки для довільних параметрів.

Обмеження програмного модуля “FALCON”:

- дозволяє працювати у наступних режимах(розмірність використаних решіток): 256, 512, 1024;
- згенерований підпис записується у окремий файл, тобто не підтримується режим роботи підпису з доповненням.

Обмеження програмного модуля “Test attacks on FALCON”:

- неможливо передати довільні параметри алгоритму до тесту, ключі генеруються та перевіряються автоматично при виборі певного режиму;
- результат роботи тестів не є очевидним з першого погляду;
- алгоритми, що використовуються не дозволяють використовувати великі значення параметрів.

5.3 Опис логічної структури

Далі буде наведено опис основних структурних частин(методів та класів) реалізованих програмних модулів.

5.3.1 Програмний модуль “FALCON”

Основними частинами цього модуля є наступні класи:

- клас FALCON(файл FALCON_oop.h та файл FALCON_oop.cpp що містить реалізацію методів) містить наступні методи:
  - FALCON(logn) що ініціалізує FALCON з певним розміром решіток(ступенем поліному що описує решітку). У даному методі logn це двійковий логарифм від n що є розмірністю решітки та дорівнює одному з трьох параметрів: 256, 512, 1024. Даний метод також ініціалізує контейнер для зберігання параметрів алгоритму та контейнер для зберігання параметрів з котрих складається декодований відкритий ключ FALCON.
  - FALCON(pubkey, privkey) що ініціалізує FALCON використовуючи файл відкритого ключа – pubkey та файл секретного ключа – privkey. Файли ключей вже містять параметр logn описаний у попередньому методі. Якщо файли були пошкоджені та не відповідають формату кодування ключів алгоритму, то ініціалізація не відбудеться та ми побачимо повідомлення про це.

- keygen() що інкапсулює у собі виклик методу котрий виконує генерацію ключів спираючись на параметри та контейнери що були створенні на етапі ініціалізації FALCON.

- sign(data) що виконує підписання даних, що містяться у data та записує підпис до контейнеру що містить параметри алгоритму. Метод інкапсулює у собі виклик методу підписання алгоритму що працює на більш низькому рівні.

- sign_file(filename) що виконує підписання файлу з іменем filename. Метод виконує зчитування даних з файлу та викликає метод sign, описаний вище, для підписання змісту цього файлу.

- write_sign_to_file(signature) що виконує запис згенерованого підпису до файлу з іменем signature. Підпис записується у кодованому форматі. Тому якщо файл підпису було пошкоджено ми отримаємо помилку про некоректний підпис. Метод записує підпис, що був поміщений у контейнер.

- write_keys_to_file(pubkey, privkey) що виконує запис ключей у відповідні файли. Ключі записуються у кодованому форматі. Першим символом у файлі йде параметр logn, що зроблено для полегшення ініціалізації процедури зчитування ключів. Крім того цей параметр закодований у деяких місцях у файлі.

- read_keys_from_file(pubkey, privkey) що виконує процедуру зчитування ключей з відповідних файлів та виконує перевірку коректності кодування.

- read_sign_from_file(file) що виконує зчитування підпису з файлу з отриманою назвою. Підпис зчитується та декодується для перевірки порушення цілісності файлу.

- verify(data, signature) що виконує перевірку переданого підпису з точки зору коректності кодування та з точки зору коректності підпису для отриманих даних з файлу з іменем data. Метод інкапсулює у собі виклик методу алгоритму що працює на більш низькому рівні. Для

перевірки підпису використовується відкритий ключ що міститься у контейнері з параметрами алгоритму. Цей ключ отримується при ініціалізації FALCON.

- verify(data, signature, pubkey) що виконує ту ж саму процедуру що й verify(data, signature) але для цього використовується ключ з файлу pubkey. Виконує ініціалізацію контексту алгоритму використовуючи файл pubkey та виконує перевірку підпису використовуючи цей контекст.

- клас FALCON_ctx(файл FALCON_ctx.h та файл FALCON_ctx.cpp що містить реалізацію методу ініціалізації контейнеру) це клас контейнер для зберігання параметрів алгоритму таких як відкритий ключ, секретний ключ, розміри параметрів, згенерований підпис та його довжина, тимчасове сховище для будь-яких параметрів що необхідні у процесі роботи алгоритму та довжина цього сховища, а також logn основний параметр алгоритму на базі якого визначаються довжини усіх елементів системи.

- структура FALCON_key_params(файл falcon.h) містить параметри відкритого ключа алгоритму у декодованому стані. Ці параметри подані у розгорнутому вигляді у першому розділі.

Також цей програмний модуль містить багато файлів у яких містяться методи та структури для роботи алгоритму на більш низькому рівні. Крім цього ці файли містять модулю для тестування швидкості роботи алгоритму та коректності роботи окремих методів. Також у них містяться методи кодування та декодування параметрів алгоритму. Описані вище класи та їх методи інкапсулюють у собі виклик методів описаних у цих файлах. Усі ці файли можна побачити у доданому до роботи проекті програми.

5.3.2 Програмний модуль “Test attacks on FALCON”

Програмний модуль складається з чотирьох файлів та виконує виклик певних методів с бібліотеки NTL[37], що була модифікована для підтримки алгоритму Self-dual BKZ. Модуль містить у собі наступні методи:

- test_with_falcon(tests_count) що виконує певну кількість викликів, котра дорівнює значенню що передається, до алгоритму тестування. Метод є проміжним для забезпечення комфортного інтерфейсу роботи з модулем. Метод виконує виклик наступного кроку для використовуючи передане значення кількості тестів та різні значення розмірності FALCON від n=32 до n=1024.

- test_with_falcon_by_size(logn, tests_count) що виконує виклик наступніх кроків тестування. Метод послідовно виконує виклик алгоритмів тестування з використанням різних алгоритмів приведення решітки. Так як метою тестування є отримання задовільних результатів, а не порівняння роботи цих алгоритмів, для кожного виклику наступного кроку тестування проводиться повторна генерація ключів FALCON.

- test_lll_falcon(FALCON* f, number) що застосовує алгоритм приведення базису решітки LLL до об’єкта f що містить у собі згенеровану пару ключів FALCON та параметри що необхідні для проведення процедури. Параметр number застосовується для позначення імені файлу до якого будуть записані результати проведення зменшення базису решітки. Виконує виклик стандартного методу з бібліотеки NTL.

- test_bkz_falcon(FALCON* f, number) що застосовує алгоритм приведення базису решітки BKZ до об’єкта f що містить у собі згенеровану пару ключів FALCON та параметри що необхідні для проведення процедури. Параметр number застосовується для позначення імені файлу до якого будуть записані результати проведення зменшення базису решітки. Виконує виклик методу з бібліотеки NTL, що був дещо модифікований для покращення результату.

- test_dbkz_falcon(FALCON* f, number) що застосовує алгоритм приведення базису решітки Self-dual BKZ до об’єкта f що містить у собі згенеровану пару ключів FALCON та параметри що необхідні для проведення процедури. Параметр number застосовується для позначення імені файлу до якого будуть записані результати проведення зменшення базису решітки.

Виконує виклик методу з бібліотеки NTL, що був імплементований згідно з алгоритмом 4.1 та виконує виклик алгоритму подвійного перерахування, шо описаний алгоритмом 4.2.

Наведені вище алгоритми дозволяють застосувати алгоритми приведення базису решітки до відкритого ключа ЕЦП FALCON.

Окрім цього програмний модуль надає можливість перевірити коректність роботи алгоритмів LLL та BKZ. Для ініціалізації цього тесту необхідним є задати розміри матриці, що утворює решітку та модуль коефіцієнтів поліному.

5.4 Технічні засоби, що використовуються

Програма не накладає обмежень на технічні засоби, що використовуються. Технічним вимогам програми задвольняє ПК з мінімальними характеристиками. Алгоритм FALCON показує гарний результати швидкості при використанні на будь-якому сучасному комп’ютері.

Програмний модуль тестування більш вибагливий до характеристик системи на якій його запускають, через те, що алгоритми приведення решіток виконують велику кількість операцій для отримання результату. Швидкість отримання результату залежить не лише від параметрів, що були надані алгоритму, а й від потужності комп’ютера.

5.5 Виклик та завантаження програми

Обидва програмних модуля об’єднані спільним інтерфейсом, що дозволяє переключатися між модулями не перезапускаючи програму або запускаючи будь-який інший файл на виконання.

Для запуску програми-інтерфейсу:

1) Програма запускається з жорсткого диску.

2) Виконуваний файл – FALCON_interface.exe

5.6 Вхідні та вихідні дані

Взаємодія користувача з програмою відбувається за допомогою консольного інтерфейсу, вхідні дані надходять або безпосередньо через елементи інтерфейсу або через файли, обрані через нього. Вихідні дані можуть бути записані у файл, ім’я котрого користувач задає самостійно.

Користувачу надається можливість обрати спосіб ініціалізації структури FALCON. Він може ініціалізувати структуру згенерувавши нову пару ключів або ж обрати файли ключів для ініціалізації структури.

Для процедури створення підпису користувачу необхідно ініціалізувати структуру одним з вище перерахованих способів.

Для процедури перевірки підпису існує два варіанти роботи. Перший варіант це перевірити підпис обравши файл даних та файл підпису з файлової системи. Другий варіант це перевірити підпис обравши файл даних, файл підпису та файл відкритого ключа, який користувач хоче використовувати для перевірки підпису.

Також користувач має можливість записати у файл згенеровану пару ключів або згенерований підпис. Для цього необхідно задати відповідні імена файлів у які будуть записані обрані дані.

Програмний модуль тестування не потребує введення даних окрім кількості тестів, що будуть проведені у випадку тестування з використанням параметрів структури FALCON. Вихідними даними є повідомлення на екрані про результат проходження тесту та файл, що містить результати обчислення приведеного базису решітки.

Перевірка роботи самих алгоритмів приведення базису потребує більше вхідних параметрів. Користувачу необхідно ввести розмірності решітки, базис котрої він хоче привести, та модуль за яким проходять перетворення. Результат роботи буде виведений на екран у тому ж вигляді що виводиться у файл.

Файли підписів не зберігають параметри криптосистеми і можуть бути розділені на складові лише з використанням відповідного ключа.

В процесі роботи можуть з’являтися додаткові інформаційні повідомлення.

5.7 Інструкція користувача

Результуюча програм має достатньо простий та зрозумілий для користувача інтерфейс.

Після запуску програми-інтерфейсу перед користувачем відкривається консоль з пропозицією обрати одну з трьох опцій (рис. А.1):

1) Ініціалізувати FALCON з новою парою ключів.

2) Ініціалізувати FALCON використовуючи існуючі ключі.

3) Перейти до роботи з модулем тестування.

Назвемо цей екран – “головний екран програми”. Це надасть нам можливість робити посилання на цей екран.

При виборі першої опції користувачу надається можливість обрати розмірність параметрів FALCON (рис. А.2). Обрати можна один з трьох розмірів – 256, 512 або 1024. При спробі ввести інший параметр користувач побачить повідомлення про некоректно введенні дані та програма запропонує ввести параметри ще один раз.

При виборі другої опції користувачу надається можливість ввести імена файлів з відкритим та закритим ключами алгоритму (рис. А.3). Якщо обрані файли містять пошкоджені або невірні ключі користувач отримає повідомлення про некоректні дані при подальшій роботі з ними.

При успішній ініціалізації алгоритму перед користувачем з’являється екран роботи з основними методами алгоритму (рис. А.4). Надається можливість обрати одну з п’яти опцій:

1) Підписати файл. При виборі цієї опції користувачу пропонується ввести ім’я файлу що підписується. Після успішного завершення процедури підпису виводиться повідомлення про завершення процедури та надається можливість одразу записати підпис у файл передавши ім’я файлу для

підпису (рис. А.5).

2) Записати ключі у файли. При виборі цієї опції користувачу пропонується ввести імена файлів спочатку для відкритого ключа, а потім для секретного ключа. Після завершення процедури запису буде виведено повідомлення про успішне завершення процедури запису (рис. А.6).

3) Записати підпис у файл. При виборі цієї опції користувачу пропонується ввести ім’я файлу для підпису. Після завершення процедури запису буде виведено повідомлення про успішне завершення процедури запису (рис. А.7).

4) Перевірити підпис. При виборі цієї опції користувачу пропонується ввести імена файлів спочатку файлу даних, що були підписані, а потім файлу підпису. Після завершення процедури перевірки буде виведено повідомлення про успішне завершення процедури перевірки (рис. А.8). Користувач отримає повідомлення про коректність або некоректність підпису. У випадку пошкодження файлу підпису виведеться повідомлення про некоректність підпису.

5) Перевірити підпис використовуючи файл відкритого ключа. При виборі цієї опції користувачу пропонується ввести імена файлів спочатку файлу даних, що були підписані, потім файлу підпису та файлу відкритого ключа. Після завершення процедури перевірки буде виведено повідомлення про успішне завершення процедури перевірки (рис. А.9). Користувач отримає повідомлення про коректність або некоректність підпису. У випадку пошкодження файлу підпису або файлу ключа виведеться повідомлення про некоректність підпису.

Обравши опцію з індексом нуль користувач може перейти на головний екран програми та обрати третю опцію що переведе його на екран вибору процедури тестування (рис. А.10). Користувачу надається можливість обрати одну з двох опцій:

1) Використати параметри алгоритму FALCON. При виборі цієї опції користувачу надається можливість ввести кількість ітерацій проведення кожної

процедури приведення базису решітки (рис. А.11). Під час проведення процедури користувачу буде надаватися інформація про поточний етап. Виводяться параметри FALCON що були використані для його ініціалізації та результат проходження тестового порогу значення рангу отриманої решітки. Окрім цього результати отримані при виконанні алгоритму користувач зможе знайти у файлах, що розміщенні у каталозі з програмою.

2) Використати довільні параметри. При виборі цієї опції користувачу надається можливість ввести параметри для ініціалізації решітки для проведення процедури (рис. А.12). Під час проведення процедури користувачу буде надаватися інформація про поточний етап. Виводяться параметри FALCON що були використані для його ініціалізації та результат проходження тестового порогу значення рангу отриманої решітки. Окрім цього на екран виводяться параметри отримані при виконанні процедури.

5.8 Отримані результати проведення процедури приведення базису решітки

Отримані результати не є задовільними. Усі тести з використанням параметрів алгоритму FALCON були завершені з помилкою та були незадовільними для подальшого проведення атаки (рис. А.13).

Модуль перетворення алгоритму FALCON q (також названий модулем коефіцієнтів для поліномів) є занадто великим для проведення цієї атаки. Як видно у таблиці 5.1 навіть при малому n результат є незадовільним для усіх алгоритмів приведення базису решітки, що були використані.

Окрім цього у таблиці 5.1 приведені такі параметри, як ранг отриманого базису та детермінант отриманої матриці, що й є базисом. Як можна побачити, незважаючи на задовільний детермінант ранг матриці після процедури зменшення є надто великим та не підходить для подальшого аналізу.

Була проведена спроба використати більш малий параметр q але ця спроба завершилася невдачею. Отримані базиси задовольняли умовам але

подальший аналіз показував, що використовуючи ці базиси неможливо тримати досить короткі вектори для відновлення секретного ключа алгоритму FALCON.

Окремо хотілося б відзначити, що деякі з тестів з використанням дійсного параметру q завершувались успіхом для розмірності n=32. Ці результати також не були задовільними через те, що отримані при подальшому аналізі вектори не могли подолати величину кореневого фактору Ерміта, що є задовільним для подальшої атаки.

Далі наведена таблиця з результатами однієї з успішних ітерації тесту.

Таблиця 5.1 — Результати роботи алгоритмів приведення базису

№	n	Алгоритм	Ранг	Визначник	Результат	фактор Ерміта
1	32	LLL	8	>100	Failed	-
		BKZ	6	1	Passed	1,1
		DBKZ	5	1	Passed	1,12
	64	LLL	12	>100	Failed	-
		BKZ	10	>100	Failed	-
		DBKZ	6	1	Passed	0,95
	128	LLL	15	1	Failed	-
		BKZ	8	>100	Failed	-
		DBKZ	14	1	Failed	-
	256	LLL	18	>100	Failed	-
		BKZ	17	>100	Failed	-
		DBKZ	18	>100	Failed	-
	512	LLL	13	>100	Failed	-
		BKZ	12	>100	Failed	-
		DBKZ	18	1	Failed	-
	1024	LLL	21	1	Failed	-
		BKZ	15	>100	Failed	-
		DBKZ	19	1	Failed	-

Як можна побачити з таблиці 5.1 навіть при задовільних результатах приведення базису, кореневий фактор Ерміта не зміг подолати планку що дорівнює 1,31 для відновлення секретного ключа алгоритму FALCON. Нагадаю, що для знаходження кореневого фактору було використано вектори отримані за допомогою алгоритму 4.2 описаному у розділі 4.4. Алгоритм Self-dual BKZ частіше за інші показував задовільні результати але вони також не підходили для подальшого аналізу.

ВИСНОВКИ

Підписи на базі алгебраїчних решіток є важливим класом постквантових криптоалгоритмів. Стійкість усіх ЕЦП цього класу базується в першу чергу на складності вирішення задачі знаходження найкоротшого вектору решітки. Електронний цифровий підпис FALCON є найбільш сучасними та перспективними підписом на баз алгебраїчних решіток. На даний час алгоритм ЕЦП FALCON є фіналістом третього раунду NIST PQC [42] та є одним з двох основних кандидатів на стандартизацію.

У даній роботі були коротко описані можливі атаки на алгоритм електронного цифрового підпису FALCON. Наведено відомі варіанти алгоритмів приведення базису решітки. Дані алгоритми є основною складовою однієї з найефективніших атак на алгоритм FALCON типу «відновлення ключа». Наведено порівняння раніше ефективних алгоритмів, а також теоретично розглянуто один з найсучасніших алгоритмів успадковує ідеї BKZ і вважається найефективнішим на даний момент серед алгоритмів приведення базису решітки алгоритм Self-Dual BKZ (DBKZ). Основна перевага полягає в тому, що в порівнянні з найефективнішим варіантом LLL-алгоритму (Slide) він здатний видавати теж якість вихідного базису за менший час при рівному розмірі блоку, а також здатний працювати з блоками в два рази більшими ніж пропонує Slide. Подано докладний опис алгоритму Self-dual BKZ з наведенням особливостей програмної реалізації. Поданий алгоритм реалізований та протестований у рамках бібліотеки NTL.

Було розроблено програмний модуль для симуляції атаки типу відновлення ключа з використанням алгоритмів приведення базису решітки. Результати роботи програми показують, що сучасний стан алгоритмів приведення решіток ще далекий від того, що потребується для проведення успішної атаки на алгоритм електронного цифрового підпису FALCON. Майже

усі ітерації атаки були провалені на першому етапі. Ті результати, що пройшли перший етап не були задовільними для подальшого проведення атаки.

З цього можна зробити висновок, що атаки типу відновлення ключа, а з нею й атаки направлені на підробку підпису не є можливими для реалізації стосовно FALCON, через те, що майже усі вони базуються на використанні алгоритму приведення базису решітки. Отже, як показали дослідження, алгоритм FALCON є стійким до відомих криптографічних атак.

ПЕРЕЛІК ПОСИЛАНЬ

1. Falcon: Fast-Fourier Lattice-based Compact Signatures over NTRU. Specifications v1.0. Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Prest, Thomas Ricosset, Gregor Seiler, William Whyte, Zhenfei Zhang. https://falcon-sign.info/falcon.pdf, дата звертання – 10.10.2020

2. Craig Gentry, Chris Peikert, and Vinod Vaikuntanathan. Trapdoors for hard lattices and new cryptographic constructions. In Richard E. Ladner and Cynthia Dwork, editors, 40th ACM STOC, pages 197–206, Victoria, British Columbia, Canada, May 17–20, 2008. ACM Press.

3. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange - A new hope. In 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016., pages 327–343, 2016.

4. Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Rosario Gennaro and Matthew J. B. Robshaw,editors, CRYPTO 2015, Part I, volume 9215 of LNCS, pages 43–62, Santa Barbara, CA,USA, August 16–20, 2015. Springer, Heidelberg, Germany/

5. Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions - cryptanalysis of some FHE and graded encoding schemes. In Matthew Robshaw and Jonathan Katz, editors, CRYPTO 2016, Part I, volume 9814 of LNCS, pages 153–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

6. A. K. Lenstra, H. W. Lenstra, Jr., and L. Lov´asz. Factoring polynomials with rational coefficients. Mathematische Annalen, 261:513–534, 1982.

7. S. Khot. Hardness of approximating the shortest vector problem in lattices. Journal of the ACM, 52(5):789–808, Sept. 2005. Preliminary version in FOCS 2004.

8. M. Albrecht, D. Cad´e, X. Pujol, and D. Stehl´e. fplll-4.0, a floating-point LLL implementation. Available at http://perso.ens-lyon.fr/damien.stehle.

9. N. Gama and P. Q. Nguyen. Finding short lattice vectors within Mordell's inequality. In Proceedings of STOC, pages 207–216. ACM, May 2008.

10. G. Hanrot, X. Pujol, and D. Stehl´e. Analyzing blockwise lattice algorithms using dynamical systems. In Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings, pages 447–464, 2011.

11. Daniele Micciancio and Michael Walter. Practical, predictable lattice basis reduction. InMarc Fischlin and Jean-Sébastien Coron, editors, EUROCRYPT 2016, Part I, volume 9665 of LNCS, pages 820–849, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg,Germany.

12. Jeffrey Hoffstein, Nick Howgrave-Graham, Jill Pipher, Joseph H. Silverman, and William Whyte. NTRUSIGN: Digital signatures using the NTRU lattice. In Marc Joye, editor, CT-RSA 2003, volume 2612 of LNCS, pages 122–140, San Francisco, CA, USA, April 13– 17, 2003. Springer, Heidelberg, Germany.

13. Oded Goldreich, Shafi Goldwasser, and Shai Halevi. Public-key cryptosystems from lattice reduction problems. In Burton S. Kaliski Jr., editor, CRYPTO'97, volume 1294 of LNCS, pages 112–131, Santa Barbara, CA, USA, August 17–21, 1997. Springer, Heidelberg, Germany.

14. Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In Serge Vaudenay, editor, EUROCRYPT 2006, volume 4004 of LNCS, pages 271–288, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

15. Léo Ducas and Phong Q. Nguyen. Learning a zonotope and more: Cryptanalysis of NTRUSign countermeasures. In Wang and Sako, pages 433–450.

16. Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In Kenneth G. Paterson, editor,EUROCRYPT 2011, volume 6632 ofLNCS, pages 27–47, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

17. Léo Ducas, Vadim Lyubashevsky, and Thomas Prest. Efficient identity-based encryption over NTRU lattices. In Palash Sarkar and Tetsu Iwata, editors, ASIACRYPT 2014, Part II, volume 8874 of LNCS, pages 22–41, Kaoshiung, Taiwan, R.O.C., December 7– 11, 2014. Springer, Heidelberg, Germany.

18. Léo Ducas and Thomas Prest. Fast fourier orthogonalization. In Sergei A. Abramov, Eugene V. Zima, and Xiao-Shan Gao, editors, Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation, ISSAC 2016, Waterloo, ON, Canada, July 19-22, 2016, pages 191–198. ACM, 2016.

19. Dan Boneh, Özgür Dagdelen, Marc Fischlin, Anja Lehmann, Christian Schaffner, and Mark Zhandry. Random oracles in a quantum world. In Dong Hoon Lee and Xiaoyun Wang, editors, ASIACRYPT 2011, volume 7073 of LNCS, pages 41–69, Seoul, South Korea, December 4–8, 2011. Springer, Heidelberg, Germany.

20. Chris Peikert. An efficient and parallel Gaussian sampler for lattices. In Tal Rabin, editor, CRYPTO 2010, volume 6223 of LNCS, pages 80–97, Santa Barbara, CA, USA, August 15–19, 2010. Springer, Heidelberg, Germany.

21. Philip N. Klein. Finding the closest lattice vector when it’s unusually close. In David B. Shmoys, editor,11th SODA, pages 937–941, San Francisco, CA, USA, January 9–11, 2000. ACM-SIAM.

22. Phong Q. Nguyen and Oded Regev. Learning a parallelepiped: Cryptanalysis of GGH and NTRU signatures. In Serge Vaudenay, editor, EUROCRYPT 2006, volume 4004 of LNCS, pages 271–288, St. Petersburg, Russia, May 28 – June 1, 2006. Springer, Heidelberg, Germany.

23. Damien Stehlé and Ron Steinfeld. Making NTRU as secure as worst-case problems over ideal lattices. In Kenneth G. Paterson, editor,EUROCRYPT 2011, volume 6632 ofLNCS, pages 27–47, Tallinn, Estonia, May 15–19, 2011. Springer, Heidelberg, Germany.

24. Daniele Micciancio and Chris Peikert. Trapdoors for lattices: Simpler, tighter, faster, smaller. In David Pointcheval and Thomas Johansson, editors, EUROCRYPT 2012, volume 7237 of LNCS, pages 700–718, Cambridge, UK, April 15–19, 2012. Springer, Heidelberg, Germany.

25. Thomas Prest. Sharper bounds in lattice-based cryptography using the rényi divergence. IACR Cryptology ePrint Archive, 2017:480, 2017. 17, 19, 37

26. Erdem Alkim, Léo Ducas, Thomas Pöppelmann, and Peter Schwabe. Post-quantum key exchange A new hope. In 25th USENIX Security Symposium, USENIX Security 16, Austin, TX, USA, August 10-12, 2016., pages 327–343, 2016.

27. Martin R. Albrecht, Shi Bai, and Léo Ducas. A subfield lattice attack on overstretched NTRU assumptions cryptanalysis of some FHE and graded encoding schemes. In Matthew Robshaw and Jonathan Katz, editors, CRYPTO 2016, Part I, volume 9814 of LNCS, pages 153–178, Santa Barbara, CA, USA, August 14–18, 2016. Springer, Heidelberg, Germany.

28. Paul Kirchner and Pierre-Alain Fouque. Revisiting lattice attacks on overstretched NTRU parameters. In Coron and Nielsen, pages 3–26.

29. Nick Howgrave-Graham. A hybrid lattice-reduction and meet-in-the-middle attack against NTRU. In Alfred Menezes, editor, CRYPTO 2007, volume 4622 of LNCS, pages 150–169, Santa Barbara, CA, USA, August 19–23, 2007. Springer, Heidelberg, Germany.

30. Paul Kirchner and Pierre-Alain Fouque. An improved BKW algorithm for LWE with applications to cryptography and lattices. In Rosario Gennaro and Matthew J. B. Robshaw, editors, CRYPTO 2015, Part I, volume 9215 of LNCS, pages 43–62, Santa Barbara, CA, USA, August 16–20, 2015. Springer, Heidelberg, Germany.

31. Nicolas Courtois, Matthieu Finiasz, and Nicolas Sendrier. How to achieve a McEliecebased digital signature scheme. In Colin Boyd, editor, ASIACRYPT 2001, volume 2248 of LNCS, pages 157–174, Gold Coast, Australia, December 9–13, 2001. Springer, Heidelberg, Germany.

32. Eike Kiltz, Vadim Lyubashevsky, and Christian Schaffner. A concrete treatment of fiatshamir signatures in the quantum random-oracle model. Cryptology ePrint Archive, Report 2017/916, 2017. http://eprint.iacr.org/2017/916.

33. Sujoy Sinha Roy, Oscar Reparaz, Frederik Vercauteren, and Ingrid Verbauwhede. Compact and side channel secure discrete Gaussian sampling. Cryptology ePrint Archive, Report 2014/591, 2014. http://eprint.iacr.org/2014/591. 21

34. NIST. Submission requirements and evaluation criteria for the post-quantum cryptography standardization process, 2016. https://csrc.nist.gov/Projects/Post-Quantum-Cryptography, дата звертання – 15.10.2002.

35. Shi Bai, Adeline Langlois, Tancrède Lepoint, Damien Stehlé, and Ron Steinfeld. Improved security proofs in lattice-based cryptography: Using the Rényi divergence rather than the statistical distance. In Tetsu Iwata and Jung Hee Cheon, editors, ASIACRYPT 2015, Part I, volume 9452 of LNCS, pages 3–24, Auckland, New Zealand, November 30 – December 3, 2015. Springer, Heidelberg, Germany.

36. NTL: A Library for doing Number Theory, 2020, https://www.shoup.net/ntl/, дата звертання – 20.11.2020

37. FpLLL library, https://github.com/fplll/fplll, дата звертання – 20.11.2020

38. Мельникова О.А., Черныш Д.И. АНАЛИЗ АЛГОРИТМА ЭЛЕКТРОННОЙ ЦИФРОВОЙ ПОДПИСИ FALCON. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління: між. конф. Полтава, 2018. с. 56.

39. Мельникова О.А., Черныш Д.И. ПРЕИМУЩЕСТВА, ОБЕСПЕЧИВАЕМЫЕ СТРУКТУРОЙ GPV В РАМКАХ АЛГОРИТМА FALCON. Сучасні напрями розвитку інформаційно-комунікаційних технологій та засобів управління: між. конф. Харків, 2019. с. 51.

40. Мельникова О.А., Черныш Д.И. АЛГОРИТМЫ СОКРАЩЕНИЯ БАЗИСА РЕШЕТКИ В КРИПТОАНАЛИЗЕ. Сучасні напрями розвитку

інформаційно-комунікаційних технологій та засобів управління: між. конф. Харків, 2019. с. 53

41. Черниш Д.І., Янко А.Г. ОБҐРУНТУВАННЯ СТІЙКОСТІ АЛГОРИТМУ ЕЦП FALCON. Матеріали першого міжнародного науково-практичного форуму «Global Cyber Security Forum». Харків. 2019. с. 108.

42. Post-Quantum Cryptography. Round 3 Submissions. https://csrc.nist.gov/projects/post-quantum-cryptography/round-3-submissions, дата звертання – 30.11.2020