МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

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

Факультет програмної інженерії та бізнесу

Кафедра інженерії програмного забезпечення

Пояснювальна записка до дипломної роботи

магістра
(освітній ступінь)

на тему «Експериментальне дослідження квантових алгоритмів цифрового підпису»

ХАІ.603.6-95пз1.121.167329.20О

Виконав: студент 6 курсу групи № 6-95пз1
Спеціальність 121 – Інженерія програмного забезпечення
(код та найменування)
Освітня програма Хмарні обчислення та Інтернет речей
(найменування)
Павлюк І.Р.
(прізвище й ініціали студента)
Керівник Туркін І.Б.
(прізвище та ініціали)
Рецензент Ільїна І.В.
(прізвище та ініціали)

Харків – 2020

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

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

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

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

З А В Д А Н Н Я
НА ДИПЛОМНУ РОБОТУ СТУДЕНТУ

Павлюк Інні Романівні
(прізвище, ім’я, по батькові)

1. Тема дипломного проекту __*Експериментальне дослідження квантових алгоритмів цифрового підпису* _____
керівник дипломного проекту *Туркін Ігор Борисович, д.т.н., професор*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “__” ___2020 року № __
2. Термін подання студентом роботи _____
3. Вихідні дані до роботи: *квантові алгоритми цифрового підпису*
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)
провести огляд та аналіз криптографічних алгоритмів цифрового підпису з точки зору їх захищеності в умовах квантових обчислень; провести експериментальне дослідження постквантових алгоритмів цифрового підпису; розробити програмну модель постквантового алгоритму цифрового підпису та провести її тестування

5. Перелік графічного матеріалу
РПЗ – стор. 75, рисунків – 36 шт., таблиць – 5 шт., презентація – 20 слайдів.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1	Туркін І.Б., зав. каф. 603		
2	Туркін І.Б., зав. каф. 603		
3	Туркін І.Б., зав. каф. 603		

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

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

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

Студент _____ ___Павлюк І.Р. ____
(підпис) (прізвище та ініціали)

Керівник роботи _____ ___Туркін І.Б._____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту містить 75 стор., 36 рис., 1 додаток, 24 джерела.

Об'єкт дослідження - алгоритми захисту від квантових атак.

Предмет дослідження - постквантові алгоритми цифрового підпису.

Метою роботи є дослідження постквантових алгоритмів для виявлення їх слабких місць з урахуванням квантових обчислень.

Для досягнення поставленої мети необхідно вирішити ряд завдань: провести огляд та аналіз криптографічних алгоритмів цифрового підпису з точки зору їх захищеності в умовах квантових обчислень; провести експериментальне дослідження постквантових алгоритмів цифрового підпису; розробити програмну модель постквантового алгоритму цифрового підпису та провести її тестування.

Наукова новизна. Удосконалено постквантовий алгоритм цифрового підпису, який на відміну від існуючих використовує механізм число-теоретичного перетворення, що дає змогу зменшити розміру ключа для криптосистеми.

Практична значимість отриманих результатів. В результаті проведених досліджень розроблено програмну модель постквантового алгоритму цифрового підпису та експериментально доведено, що комбінований розмір підпису та публічного ключа досить конкурентноспроможний з іншими ефективними схемами на решітках.

ЦИФРОВЕ ШИФРУВАННЯ, ЕЛЕКТРОННИЙ ЦИФРОВИЙ ПІДПИС, АЛГОРИТМ ШОРА, КВАНТОВИЙ КОМП’ЮТЕР, ПОСТКВАНТОВА КРИПТОГРАФІЯ, QTESLA, СХЕМА ПІДПИСУ

ABSTRACT

Explanatory note to the master's thesis 75 pp., 36 fig., 1 app., 24 sources.

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

The subject of research is post-quantum digital signature algorithms.

The aim of the work is to study post-quantum algorithms to identify their weaknesses, taking into account quantum calculations.

To achieve this goal it is necessary to solve a number of tasks: to review and analyze cryptographic algorithms of digital signatures in terms of their security in terms of quantum computing; to conduct an experimental study of post-quantum digital signature algorithms; develop a software model of the post-quantum digital signature algorithm and test it.

Scientific novelty. The post-quantum digital signature algorithm has been improved, which, unlike the existing ones, uses a number-theoretical transformation mechanism, which allows to reduce the key size for the cryptosystem.

The practical significance of the results obtained. As a result of the research, a software model of the post-quantum digital signature algorithm was developed and it was experimentally proved that the combined size of the signature and the public key is quite competitive with other efficient schemes on lattices.

DIGITAL ENCRYPTION, ELECTRONIC DIGITAL SIGNATURE, SHORE ALGORITHM, QUANTUM COMPUTER, POSTQUANTIC CRYPTOGRAPHY, QTESLA, SIGNATURE SCHEME

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ЕЦП - Електронний цифровий підпис.

NTT - Число-теоретичне перетворення.

ВСТУП

Потенційна поява квантових комп'ютерів змусила криптографічне співтовариство шукати квантово-стійкі альтернативи класичних схем, які засновані на факторизації і завданнях дискретного логарифма.

У цій роботі досліджено сімейство схем цифрового підпису на основі решітки, під назвою qTESLA, яка об'єднує ряд недавніх зусиль щодо розробки ефективної та квантово-безпечної схеми підпису.

Безпека qTESLA побудована на так званій задачі навчання з помилками (R-LWE) [1]. Тобто реалізації схеми гарантує певний рівень безпеки, оскільки відповідні екземпляри R-LWE дають певну стійкість.

Найбільш важливі особливості qTESLA узагальнені наступним чином:

Простота. qTESLA розроблена так, щоб її було легко реалізувати, з особливим акцентом на найбільш часто використовувані функції в схемі підпису, а саме, підписи і перевірки. Зокрема, вибірка Гаусса, можливо, найскладніша частина, традиційні схеми підпису на основі решітки, придатна виключно для генерації ключів.

Реалізація. Наприклад, еталонна реалізація написана за допомогою мови C і підтримує всі набори параметрів qTESLA, складається всього з 300 рядків коду.

Основи безпеки. Безпека qTESLA забезпечується зниженням безпеки в квантової моделі випадкового оракула (QROM). [2]

Практична безпека. qTESLA полегшує реалізацію, захищену від атак на реалізацію. Наприклад, він підтримує реалізації з постійним часом (тобто реалізації, які захищені від синхронізації і атак по побічному каналу кешу, завдяки виключенню доступу до секретної пам'яті) і за своєю природою захищений від деяких простих, але потужних атак [3,4]. Більш того, дана реалізація також поставляється з вбудованим захистом від заміни ключа - KS атаки [5, 6] (Атаки з дублюванням ключа підпису (DSKS)) і, таким чином, покращено безпеку в розрахованому на багато користувачів, режимі. [7].

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Мало хто з нас звертав увагу на невеликій символ замочка, що з'являється в наших веб-браузерах кожен раз, коли ми заходимо на сайт інтернет-магазину, відправляємо і отримуємо емейли, перевіряємо наш банківський рахунок або кредитну карту. Однак він сигналізує про те, що онлайн-сервіси використовують HTTPS, веб-протокол, який шифрує дані, які ми відправляємо по інтернету, і ті відповіді, що ми отримуємо. Ця та інші форми шифрування захищають різні електронні комунікації, а також такі речі, як паролі, цифрові підписи та історії хвороб.

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

1.1 Цифрове шифрування

Існують два основних типи шифрування. Симетричне шифрування [8] вимагає, щоб у відправника і одержувача були в наявності ідентичні цифрові ключі для шифрування і розшифровки даних, а асиметричне - або шифрування з публічним ключем - використовує публічно доступний ключ, що дозволяє людям зашифровувати повідомлення для одержувача, який в свою чергу одноосібно володіє приватним ключем, що дозволяє їх розшифровувати .

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

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

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

1.2 Електронний цифровий підпис

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

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

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

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

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

1.3 Загроза шифруванню від квантових комп'ютерів

Оскільки вони можуть допомогти хакерам набагато швидше пробиратися через алгоритмічні потаємні ходи. На відміну від класичних комп'ютерів, що використовують біти, здатні приймати лише значення 1 або 0, квантові машини використовують кубіти, здатні одночасно представляти різні можливі стани, проміжні між 0 і 1 - це явище називається суперпозицією. Вони також можуть впливати один на одного на відстані завдяки такому явищу, як заплутаність [10].

Завдяки цьому явищу додавання декількох додаткових кубітів може привести до експоненціальних стрибків обчислювальної потужності. Квантова машина з 300 кубитами здатна представляти більше значень, ніж кількість атомів в спостережуваного Всесвіту. Припускаючи, що квантові комп'ютери зможуть подолати деякі властиві їм обмеження, що стосуються швидкодії, коли-небудь їх можна буде використовувати для перевірки всіх можливих варіантів криптографічного ключа за відносно недовгий час.

Хакери також, швидше за все, спробують використовувати на шкоду алгоритми оптимізації певних завдань. Один з таких алгоритмів, опублікований Гровером з AT & T Bell Labs, допомагає квантовим комп'ютерам набагато швидше

шукати варіанти. Інший алгоритм, опублікований в 1994 році Пітером Шором, тоді також працювали в Bell Labs, а тепер професором в MIT, допомагає квантовим комп'ютерам неймовірно швидко знаходити прості множники цілих чисел.

Алгоритм Шора [11] загрожує таким системам з публічним ключем, як RSA, чий математичний захист, зокрема, залежить від того, наскільки складно провести зворотний інжиніринг результату перемноження дуже великих простих чисел (розкладання на множники). У звіті з квантових обчислень, опублікованому в минулому році національною академією наук, інженерної справи і медицини США, передбачається, що потужний квантовий комп'ютер, на якому працює алгоритм Шора, зможе зламувати 1024-бітові варіанти RSA менш, ніж за день.

Дослідження національних академій стверджує, що для того, щоб представляти реальну загрозу, квантових комп'ютерів знадобиться набагато більше обчислювальної потужності, ніж є у найкращих з них на сьогодні [12].

Однак, рік, в якому квантовий злом кодів стане серйозним головним болем - який деякі дослідники безпеки охрестили Y2Q - може підібратися несподівано швидко. У 2015 році дослідники зробили висновок, що для досить швидкого злому 2048-бітного RSA шифрування квантовому комп'ютера знадобиться мільярд кубітів. У більш сучасноій роботі вказується, комп'ютер з 20 млн кубітів зможе впоратися з цим завданням всього за 8 годин.

Це знаходиться далеко за межами можливостей найпотужніших з сьогоднішніх комп'ютерів, що володіють всього 128 кубитами. Але прогрес квантових обчислень непередбачуваний. Без криптографічного захисту, що враховує квантові обчислення, сервіси - від військового обладнання, до фінансових транзакцій і комунікацій - підвержені загрозам атаки з боку хакерів, які отримали доступ до квантових комп'ютерів.

Будь-яке підприємство чи уряд, котре розраховує зберігати данні кілька десятиліть, має вже зараз замислитися над тим, які ризики несе в собі нова технологія, оскільки використовується ними сьогодні шифрування може бути зламано в майбутньому. На перекодування величезних обсягів історичних даних в

більш надійний вид можуть піти роки, тому краще буде застосовувати надійне кодування сьогодні. Звідси виникає запит на постквантовую криптографію.

1.5 Постквантова криптографія

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

Одна з ліній оборони - збільшення розмірів цифрових ключів для значного збільшення кількості варіантів, в яких потрібно буде вести пошук перебором. Наприклад, просте подвоєння розміру ключа з 128 до 256 біт збільшить кількість можливих варіантів в чотири рази, які доведеться перебрати квантовій машині, що використовує алгоритм Гровера [13].

Ще один підхід включає використання більш складних функцій з потайним входом, таких, щоб з ними важко було впоратися навіть потужному квантовому комп'ютера, виконуючому алгоритм Шора. Дослідники працюють над широким спектром підходів, включаючи такі екзотичні, як криптографія на решітках і протокол обміну ключами з використанням суперсінгулярного ізогенна [14].

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

Критична важливість цього завдання пов'язана з тим, що технології шифрування глибоко впроваджені в безліч різних систем, тому на їх переробку і впровадження нових алгоритмів потрібно дуже багато часу. У дослідженні національних академій від минулого року відзначено, що на повне позбавлення від одного широко розповсюдженого криптографічного алгоритму, який опинився

вразливим, пішло більше 10 років. З огляду на швидкість розвитку квантових комп'ютерів, можливо, у світу залишилося не так вже й багато часу, щоб розібратися з цією новою проблемою безпеки.

Ось чому дослідження пост квантових алгоритмів цифрового підпису та їх порівняльний аналіз дуже важливий. В роботі досліджується алгоритм цифрового підпису QTesla

1.6 Порівняння з аналогами

У таблиці (1.1) qTESLA порівнюється з репрезентативними сучасними схемами постквантового підпису з точки зору бітової безпеки, розмірів підписів та відкритих ключів, продуктивність портативних C-посилань та типу процесору мінімально необхідного для обробки алгоритму. Якщо в літературі наведено як медіану, так і середню кількість циклів, ми повідомляємо про середнє для підписання та медіану для перевірки. Для того щоб отримати справедливі результати порівняння, ми констатуємо безпеку бітів qTESLA, Falcon та Dilithium якщо припустити ту саму модель вартості $BKZ\ 0{,}265\beta + 16{,}4 + log_2(8d)$, причому β є розміром BKZ, а d - розміром решітки.

Таблиця 1.1 – Порівняння схем постквантового підпису

Схема	Безпека (біт)	Постійний час	Розміри	Кількість циклів	CPU
qTESLA	160	так	pk: 38,432 sig: 5,664	sign: 7,122.6 verify: 2,102.3	S
FALCON-512	158	ні	pk: 897 sig: 617	sign: 1,368.5 verify: 95.6	S
Dilithium	160	так	pk: 1,472 sig: 2, 701	sign: 2,035.9 verify: 375.7	S
SPHINCS+	128	так	pk: 32 sig: 16,976	sign: 325,311 verify: 13,541	H

FALCON-512, єдина схема, що пропонує параметри відповідно до їх зниження рівня безпеки, має найменший розмір $(pk + sig)$ серед усіх схем постквантового підпису, приведених для порівняння. Однак у Falcon є деякі недоліки через високу складність. Ця схема спирається на дуже складний метод - відбору проб Фур'є, та вимагає арифметики з плаваючою комою, що не підтримується більшістю пристроїв. Це робить схему значно складнішою для реалізації, взагалі, та слабо-захищеною від атак бокових каналів та несправностей. Нещодавнє ефективне впровадження Порніном [15], використовує складний код емуляції з плаваючою комою для вирішення проблем переносимості, і містить кілька тисяч рядків коду C. Проте дане програмного забезпечення досі не може бути відмічене як $constant - time$, оскільки деякі його частини допускати деяку лімітовану кількість просочення.

Схеми, засновані на інших основних проблемах, таких як SPHINCS+, пропонують компактні відкриті ключі за рахунок дуже великих розмірів підписів.

Підсумовуючи, qTESLA пропонує хороший баланс між ефективністю, що супроводжується простим, компактним і надійним дизайном.

2 ПЛАНУВАННЯ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ КВАНТОВОГО АЛГОРИТМУ ЦИФРОВОГО ПІДПИСУ QTESLA

2.1 Схема підпису qTESLA

У цьому розділі ми описуємо схему підпису qTESLA, деяку з її найбільш відповідних функцій дизайну та всі параметри системи. Почнемо з опис схеми.

qTESLA параметризується такими параметрами як: $\lambda, \kappa, n, k, q, \sigma, L_E, L_S, E, S, B, d, h$ та b_{GenA}; див. таблицю (2.1) для детального опису всіх параметрів системи.

Таблиця 2.1 – Опис та межі всіх параметрів системи

Параметр	Опис	Вимоги
Λ	параметр безпеки	-
N	вимір	2^e
Σ	стандартне відхилення D_σ	-
K	№ публічних многочленів $a_1, \ldots, a_k$	-
Q	модуль	$q = 1\ mod\ 2n, q > 2B, q2^{d+1}q^{nk} \geq \lvert\Delta S\rvert \cdot \lvert\Delta L\rvert \cdot \lvert\Delta H\rvert,$ $q^{nk} \geq 2^{4\lambda+nkd}4q_S^3(q_s+q_h)^2$
K	довжина вводу / вводу функцій	$\kappa \geq \lambda$
L_E, η_E	прив'язаний у випробуванні E	$[\eta_E \cdot h \cdot \sigma]$
L_S, η_S	прив'язаний у випробуванні S	$[\eta_S \cdot h \cdot \sigma]$
S, E	параметр відхилення	$= L_S, L_E$
B	визначає інтервал випадковості під час підпису	поблизу потужності, B $\geq \frac{\sqrt[n]{M}+2S-1}{2(1-\sqrt[n]{M})}$

Кінець таблиці 2.1

D	№ округленого біта	$\mathrm{d} > \log_2(\mathrm{B}), \mathrm{d} \geq \log_2(\frac{2E+1}{1-M\frac{1}{nk}})$
H	№ ненульових записів вихідних елементів E_{nc}	$2^{\mathrm{h}} \cdot (\frac{\mathrm{n}}{\mathrm{h}}) \geq 2^{2\lambda}$
b_{Gen}	№ блоку запитані SHAKE128 для Gen	$b_{\mathrm{Gen}} \in Z > 0$

Для реалізації схеми необхідні наступні функції:

а) Псевдовипадкові функції $\mathrm{PRF}_1: \{0,1\}^{\kappa} \rightarrow \{0,1\}^{\kappa,k+3}$, що приймає як вхідний параметр seed pre-seed, який є довгим к, і відобразить його на $(k + 3)$ seed к біт кожен.

б) Хеш-функція стійкого до зіткнення $\mathrm{G}: \{0,1\}^{*} \rightarrow \{0,1\}^{320}$, які карти заданого вхідного рядка до 320-бітного рядка.

в) Псевдовипадкова функція $\mathrm{PRF}_2: \{0,1\}^{\kappa} \times \{0,1\}^{\kappa} \times \{0,1\}^{320} \rightarrow \{0,1\}^{\kappa}$, яке приймає як вхідне значення seed та випадкове значення r, кожен к біт довгий, і хеш G повідомлення m, який має розмір 320 біт, і відображає їх у к-бітний seed rand.

г) Функція генерації многочленів $a_1, \ldots, a_{\mathrm{к}}, GenA: \{0,1\}^{\kappa} \rightarrow R_q^k$ який приймає за вхід к-бітове $seed_a$ і відображає його на k многочленах $a_i \in R_q$.

д) Гауссова функція пробо-відбору $\mathrm{GaussSampler}: \{0,1\}^{\kappa} \times Z \rightarrow R$, який приймає за вхід к-бітове $seed \in \{seed_S, seede_{e1}, \ldots, seed_{ek}\}$ і $nonce\ counter \in Z > 0$, і виводить многочлен в R, відібраний відповідно до дискретного гауссового розподілу D_{σ}.

ж) Функція кодування $Enc: \{0,1\}^{\kappa} \rightarrow \{0, \ldots, n-1\}^h \times \{-1,1\}^h$ кодує к-бітове значення c' як многочлен $c \in H_{n,h}$. Поліном c представлений у вигляді списку $pos\_list \in \{0, \ldots, n-1\}^h$ і $sign\_list \in \{-1,1\}^h$, що містить положення та ознаки його ненульових коефіцієнтів, відповідно.

з) Функція вибірки $ySampler: \{0,1\}^{\kappa} \times Z \rightarrow R_{[B]}$ вибірки многочлена $y \in R_{[B]}$, приймаючи за вхід к-бітовий seed рядок і $nonce\ counter \in Z > 0$.

к) Хеш-функція H: $R_q^k \times \{0,1\}^{320} \times \{0,1\}^{320} \rightarrow \{0,1\}^{\kappa}$. Це функція приймає як входи k многочленів $v_1, \ldots, v_k \in R_q$ і перші обчислення $[v_1]_{\text{M}}, \ldots, [v_k]_M$. Потім результат хеширують разом з хешем $G(m)$ для заданого повідомлення m і хеша $G(t_1, .., t_k)$ до рядка κ біт long.

л) Функція перевірки коректності $checkE$, яка отримує поліном помилки e як вхід і відхилення, якщо $\sum_{k=1}^{h} max_k(e)$ більше, ніж деякий зв'язаний L_E. Перевірка функції гарантує правильність схеми підпису, забезпечуючи $\|e_i c\|_\infty \leq E \in \{L_E, 2L_E\}$ для $i = 1, \ldots, k$ під час генерації ключів.

м) Функція перевірки спрощення $checkS$, яка отримує таємний многочлен s як вхід і відхиляє його, якщо $\sum_{k=1}^{h} max_k(s)$ більше, ніж деякі пов'язані L_S. $checkS$ забезпечує $\|cs\|_\infty \leq S \in \{L_S, 2L_S\}$, що використовується для спрощення зниження рівня безпеки.

Зараз ми можемо описати алгоритми qTESLA для генерації ключів, підписання та перевірка.

Генерація ключів. По-перше, публічні многочлени $a_1, \ldots, a_k$ генеруються рівномірно випадково по R_q шляхом розширення $seed_a$ за допомогою PRF_1. Тоді секретний поліном s відбирається з дискретним розподілом Гаусса D_σ. Цей многочлен повинен відповідати вимозі в $checkE$. Аналогічна процедура вибірки таємних поліномів помилок $e_1, \ldots, \text{e}_{\text{к}}$. У такому випадку ці поліноми повинні виконати перевірку правильності в $checkE$.

Для генерації псевдовипадкових бітових рядків під час дискретного гауссова вибірки відповідне значення від $\{seed_s, seed_{e1}, \ldots, seed_{ek}\}$ використовується як $seed$, і а лічильник використовується як $nonce$ для забезпечення поділу домену між різними викликами до вибірки. Відповідно, цей лічильник ініціалізується на 1 і потім збільшується на 1 після кожного виклику до проби Гаусса. Нарешті, відкритий ключ pk складається з $seed_a$ та многочленів $t_i = a_i s + e_i \; mod \; q$ при $i = 1, \ldots, k$, і секретний ключ sk складається з $s, e_1, \ldots, e_k$, $seed_a$ і $seed_y$, і хеш $g = G(t_1, .., t_k)$. Всі $seed$, необхідні під час генерації ключа, генеруються компанією розширення pre-seed за допомогою PRF_1.

Генерація підписів. Щоб підписати повідомлення m, спочатку многочлен $y \in R_{[B]}$ вибирається рівномірно випадково. З цією метою лічильник, ініціалізований на одиниці, використовується як $nonce$, так і випадковий рядок - $rand$, обчислений як $PRF_2(\text{seed}_y, \text{r}, \text{G(m)})$ з seed_y, випадковий рядок r, і дайджест $G(m)$ повідомлення m, використовується в якості $seed$. Лічильник використовується для забезпечення поділу домену між різними викликами до випадку y. Відповідно, вона збільшується на 1 щоразу, коли алгоритм перезапускається якщо будь-який із тестів на безпеку чи правильність не зможе обчислити дійсну підпис. Далі, $seed_a$ розширюється для отримання многочленів $a_1, \ldots, a_k$ які потім використовуються для обчислення поліномів $v_i = a_i y \; mod \pm q$ для $i = 1, \ldots, k$. Після цього хеш-функція обчислює $[v_1]_M, \ldots, [v_k]_M$ і хешує їх разом із дайджестами $G(m)$ і g для отримання c'. Потім це значення детерміновано відображається на генерований псевдовипадовий многочлен $c \in H_{n,h}$, кодований у вигляді двох масивів $pos\_list \in \{0, \ldots, n-1\}^h$ і $sign\_list \in \{-1, 1\}^h$ що представляють положення та ознаки ненульових коефіцієнтів c відповідно. Для того, щоб потенційний підпис $(z \leftarrow sc + y, c')$ для повернення алгоритмом підпису, потрібно пройти захист і перевірку правильності, які описані далі.

Застосовується перевірка безпеки, яка також називається вибіркою відхилення, переконайтеся, що підпис не просочує жодної інформації про секрет. Це реалізується, перевіривши, що $z \notin R_{[B-S]}$. Якщо перевірка не вдається, алгоритм відкидає поточну пару (z, c') і повторює всі кроки, починаючи з вибірки y. В іншому випадку алгоритм продовжується з перевіркою правильності. Перевірка правильності забезпечує правильність схеми підпису, тобто гарантує, що кожен дійсний підпис, що генерується алгоритмом підпису, приймається алгоритмом верифікації. Він реалізується за допомогою перевірки $\|[w_i]_L\|_\infty < 2^{d-1} - E$ і $\|[w_i]\|_\infty < \lfloor q \,/\, 2 \rfloor - E$. Якщо перевірка не вдається, то алгоритм відкидає поточну пару (z, c') і повторює всі кроки, що починаються вибіркою y. В іншому випадку він повертає підпис (z, c') на m.

Перевірка. Алгоритм перевірки, при введенні повідомлення m, підпис (z, c') і відкритим ключем pk, обчислює $\{pos\_list, sign\_list\} \leftarrow Enc(c')$, розширюється $seed_a$ для генерації $a_1, \ldots, a_k \in R_q$, а потім обчислює $w_i = a_i z - b_i c \; mod \pm q$ для $i = 1, \ldots, k$. Хеш-функція H обчислює $[w_1]_M, \ldots, [w_k]_M$ і хешує їх разом із дайджестами $G(m)$ і $G(t_1, \ldots, t_k)$. Якщо рядок біт в результаті попереднього обчислення відповідає рядку біта підпису c', і $z \in R_{[B-S]}$, підпис приймається; в іншому випадку він відхиляється.

Правильність qTESLA. Щоб гарантувати правильність qTESLA, він повинен відповідати підпису (z, c') повідомлення m, породженого алгоритмом генерації, що (i) $z \in R_{[B-S]}$ і що (ii) висновок хеш-функції H при підписанні такий же, як і аналогічний вихід при перевірці. Вимога (i) забезпечує перевірку безпеки під час підписання. Для забезпечення (ii) використовується перевірка правильності при підписанні. По суті, ця перевірка гарантує, що для $i = 1, \ldots, k, [a_i z - t_i c]_M = [a_i(y + sc) - (a_i s + e_i)c]_M = [a_i y - e_i c]_M = [a_i y]_M$.

Особливості дизайну. Дизайн qTESLA оснащений кількома вбудованими функціями безпеки. По-перше, публічні многочлени $a_1, \ldots, a_k$ генеруються при кожному створенні ключа, використовуючи випадкові $seed_a$. Це $seed$ зберігається як частина обох sk і pk, щоб операції підписання та перевірки могли відновити $a_1, \ldots, a_k$. Це ускладнює введення так званих – backdoors, і різко скорочує їх масштаби атак. Більше того, зберігання лише $seed$ замість повних многочленів дозволяє зберегти пропускну здатність, оскільки нам потрібен лише κ біти для зберігання $seed_a$ замість $kn\lceil log_2(q) \rceil$ біти, необхідні для представлення повного многочлена.

Для захисту від атак KS [16] ми включаємо хеш G поліномів $t_1, \ldots, t_k$ (які є частиною відкритого ключа) у секретному ключі, з метою використання цього під час операції хешування c'. Це гарантує, що, будь-яку спробу зловмисником зміни публічного ключа, буде виявлено під час перевірки значення c'.

Також $seed$, який використовується для створення випадковості y, при підписанні, створюється значення, яке є частиною секретного ключа, деяким

випадковим r, і дайджестом $G(m)$ повідомлення m. Використання $seed_y$ робить qTESLA стійким до катастрофічного виходу з ладу, генератора випадкових чисел (RNG), під час генерації нових випадкових, захищаючи від атак фіксованої випадковості наприклад, продемонстровано проти Playstation 3 від Sony [17]. Точно також випадкове значення r гарантує використання свіжої y при кожній операції підписання, яка робить підписи qTESLA вірогідними. Ймовірнісні підписи, можливо, складніше атакувати за допомогою аналізу бічних каналів. Причому свіжий y запобігає легким у здійсненні, але потужним нападам - несправностей проти детермінованих схем підпису [18, 19]. Відзначимо що використання PRF (у нашому випадку PRF_2) зменшує потребу у якісній джерело випадковості для генерування r.

Іншою особливістю дизайну qTESLA є те, що, лише дискретна гауссова вибірка найскладніша функція у багатьох схемах підписів на основі ґрат, необхідна під час генерації ключів, під час підписання та перевірки. Функції схем цифрового підпису, використовують лише дуже прості арифметичні операції, які легко здійснити. Це сприяє реалізації компактних і портативних реалізацій, які досягають високої продуктивності.

2.2 Число-теоретичне перетворення (NTT)

NTT - це спеціалізована версія дискретного перетворення Фур'є, в якій кільце коефіцієнта прийнято вважати кінцевим полем (або кільцем), що містить правильні корені. Його можна розглядати як точну версію складного DFT, уникаючи помилок округлення для точних згортків цілих послідовностей. Тоді як Гаусс мабуть використовував подібні методи вже в основні роботи для сучасного FFT алгоритму, для обчислення DFT.

Позначення та передумови. З параметрами, наведеними вище, тобто n є потужністю 2 і q просте з $q = (1 \bmod 2n)$, нехай $a = (a_0, \ldots, a_{n-1}) \in Z_q^n$, і нехай ω буде примітивний n-й корінь в Z_q, що означає, що $\omega^n \equiv 1(mod\ q)$. Пряме перетворення $\tilde{a} = NTT(a)$ визначається як $\tilde{a}_i = \sum_{j=0}^{n-1} a_j \omega^{ij} mod\ q$ для $i = 0, 1, \ldots, n-1$. Зворотне перетворення задається як $b = INTT(\tilde{a})$, де $b_i = n^{-1} \sum_{j=0}^{n-1} \tilde{a}_j\ \omega^{-ij} mod\ q$ при $i = 0, 1, \ldots, n-1$, і маємо $INTT(NTT(a)) = a$.

Як було сказано вище, NTT можна використовувати безпосередньо для виконання основної операції в криптографії, на основі R-LWE, тобто множення полінома $R_q = Z_q[X]/(X^n + 1)$. Однак застосування NTT перетворення, як описано вище, забезпечує циклічну згортку, обчислюючи $c = a \cdot b\ mod(X^n + 1)$ з двома многочлени a і b вимагають застосування NTT довжиною $2n$ і, таким чином n нулів, які слід додати до кожного входу; це ефективно подвоює вхідну довжину, а також вимагає обчислення явного модуля скорочення $X^n + 1$. Щоб уникнути цих питань, можна використовувати негативну згортку: нехай ψ буде примітивним другим коренем в Z_q таким, що $\psi^2 = \omega$, і нехай $a = (a_0, \ldots, a_{n-1}), b = (b_0, \ldots, b_{n-1}) \in Z_q^n$ - два вектори. Також визначимо $\hat{a} = (a_0, \psi a_1, \ldots, \psi^{n-1} a_{n-1})$ і $\hat{b} = (b_0, \psi b_1, \ldots, \psi^{n-1} b_{n-1})$. Негативно завернута a і b визначається як $c = (1, \psi^{-1}, \psi^{-2}, \ldots, \psi^{-(n-1))} \circ INTT(NTT(\hat{a}) \circ NTT(\hat{b}))$, де $\circ$ означає компонентне множення. Це операція задовольняє $c = a \cdot b$ в R_q, і, таким чином, вона дозволяє обчислити повне поліноміальне множення, що неявно включає модуль відновлення $X^n + 1$, без збільшення довжини входів.

2.2.1 Алгоритм метелика

Розглянемо алгоритм метелика: $\begin{bmatrix} X \\ Y \end{bmatrix} \rightarrow \begin{bmatrix} X+Y \\ W(X-Y) \end{bmatrix}$. NTT витрачає $O(N\ log\ N)$ процесорного часу на цю операцію. $O(N)$ з них має $W = 1$; ми сконцентруємося на випадку, коли $W \neq 1$. Ми припускаємо що W походить від результатів пошуку таблиці. Наша увага зосереджується на наступній проблемі: задані X, Y і W, як найбільш ефективно обчислити $X + Y$ і $W(X - Y)$?

Дозволені примітивні операції:

додавання / віднімання окремих слів (модуль β)

а) порівняння одиничних слів

б) множення одиночних слів (модуль β)

в) широке множення, тобто, задавши $U, V \in [0, \beta)$, обчислюють UV у вигляді $UV = P_1\beta + P_0$, де $P_0, P_1 \in [0, \beta)$.

Я також припускаю, що ми маємо: двохслівний поділ, тобто заданий $U \in [0, \beta^2)$ і $M \in [0, \beta)$, обчислити як $Q = \lfloor U/M \rfloor$ і $R = U - QM$

Алгоритм операції метелика:

Вхід: $X, Y, W \in [0, p)$, припустимо що, $p < \beta/2$

Вихід: $\begin{matrix} X' = X + Y\ mod\ p \\ Y' = W(X - Y)\ mod\ p \end{matrix}$

а) $X' \leftarrow X + Y$

б) $if\ X' \geq p\ then\ X' \leftarrow X' - p$ (поточний $X' = X + Y\ mod\ p$)

в) $T \leftarrow X - Y$

г) $if\ T < 0\ then\ T \leftarrow T + p$ (поточний $T = X - Y\ mod\ p$)

д) $U = U1\beta + U0 \leftarrow TW$ (широке множення)

е) $Y' \leftarrow U\ mod\ p$ (двохслівний поділ)

Це неефективно, оскільки апаратний поділ повільний.

2.2.2 Модульне множення

Як ми можемо обчислити $TW \bmod p$ ефективніше? Існує кілька відомих методів, які замінюють поділ на множення.

Основна стратегія:

а) Оцінити «коефіцієнт» Q.

б) Помножити Q на p.

в) Відняти Qp від TW, щоб отримати залишок кандидата R.

г) додаю/віднімаю невелике число, кратне p, щоб змінити залишок на стандартний інтервал $[0, p)$.

Алгоритм модульного множення:

Вхід: $T, W \in [0, p)$, припустимо що, $p < \beta/2$ попередньо обчислена $W' = \lfloor W\beta/p \rfloor$

Вихід: $R = TW \bmod p$

а) $Q \leftarrow \lfloor W'T/\beta \rfloor$ (більша частина продукту $W'T$)

б) $R \leftarrow (WT - Qp) \bmod \beta$ (дві менші частини продукту)

в) $if\ R \geq p\ then\ R \leftarrow R - p$

Примітка: W інваріантний. Це розумно для NTT, оскільки кожна трансформація використовує ті самі корені.

W'- масштабоване наближення до $W/$p.

Q- наближення до WT/p.

$WT - Qp \in [0, 2p)$. Таким чином, R, обчислена у рядку 2, є саме $WT - Qp$. Останній рядок налаштовує залишок на $[0, p)$.

Алгоритм метелика з модульним множенням

Вхід: $X, Y, W \in [0, p)$, припустимо що, $p < \beta/2$ попередньо обчислена $W' = W\beta/p$

Вихід: $X' = X + Y \bmod p, Y' = W(X - Y) \bmod p$

а) $X' \leftarrow X + Y$

б) $if\ X' \geq p\ then\ X' \leftarrow X' - p$

в)

г) $T \leftarrow X - Y$

д) $if\ T < 0\ then\ T \leftarrow T + p$

е) $Q \leftarrow \lfloor W'T/\beta \rfloor$

ж) $Y' \leftarrow (WT - Qp)\ mod\ \beta$

ж) $if\ Y' \geq p\ then\ Y' \leftarrow Y' - p$

2.2.3 Видалення «if» кроків

Наша мета: видалити якомога більше «if» кроків "if (деяка умова) $then\ Z \leftarrow Z \pm p$".

Кожне «if» вимагає декількох інструкцій на машині: a- умовний хід та кілька інших інструкцій щодо його налаштування. Ці «if» можуть становити значну частку загального часу виконання. Особливо на сучасних процесорах з дуже швидкими множниками!

Одне «if» легко зняти. В алгоритмі Shoup для обчислення $TW\ mod\ p$ ми припустили $T \in [0, p)$. Але насправді алгоритм прекрасно працює для будь-якого $T \in [0, \beta)$. Тож ми можемо просто пропустити «if» для T.

Як щодо останнього коригування для Y'? Коли виходи метелика лежать у $[0, 2p)$. Розкладіть алгоритм, щоб входи могли лежати у $[0, 2p)$. Іншими словами, працює весь алгоритм FFT «Неканонічні залишки». У кожного елемента Fp є два можливих репрезентації, одна в $[0, p)$ і одна в $[p, 2p)$. За бажанням, кінцевий прохід зменшує вихід на $[0, p)$. Для роботи цієї схеми нам потрібен $p < \beta/4$.

Алгоритм метелика з модульним множенням та відсутніми двома «if»

Вхід: $X, Y \in [0, 2p), W \in [0, p)$, припустимо що, $p < \beta/4$ попередньо обчислена $W' = \lfloor W\beta/p \rfloor$

Вихід: $X' \equiv X + Y\ mod\ p, Y' \equiv W(X - Y)\ mod\ p$
$X', Y' \in [0, 2p)$

а) $X' \leftarrow X + Y$

б) $if\ X' \geq 2p\ then\ X' \leftarrow X' - 2p$

в) $T \leftarrow X - Y + 2p$

г) $Q \leftarrow \lfloor W'T/\beta \rfloor$

д) $Y' \leftarrow (WT - Qp)\ mod\ \beta$

2.2.4 Зворотній алгоритм метелика

Розглянемо "зворотний алгоритм метелика" $\begin{bmatrix} X \\ Y \end{bmatrix} \rightarrow \begin{bmatrix} X + WY \\ X - WY \end{bmatrix}$. Це обертання звичайного алгоритму метелика (після заміни W на W^{-1}, і ділення на 2). Зворотний алгоритм метелика природним чином з'являється в оберненій FFT алгоритму. Можна також реалізувати перемотку FFT за допомогою оберненого метелика шляхом переходу від децимації за частотою до децимації в часі. Подібна хитрість стосується і зворотного метелика, але зараз ми використовуємо представники в $[0, 4p)$:

Зворотній алгоритм метелика з модульним множенням та відсутніми двома «if»

Вхід: $X, Y \in [0, 4p), W \in [0, p)$, припустимо що, $p < \beta/4$ попередньо обчислена $W' = \lfloor W\beta/p \rfloor$

Вихід: $X' \equiv X + Y\ mod\ p, Y' \equiv W(X - Y)\ mod\ p$
$\mathrm{X'}, \mathrm{Y'} \in [0, 4\mathrm{p})$

а) $if\ X \geq 2p\ then\ X \leftarrow X - 2p$ (поточний $\mathrm{X} \in [0, 2\mathrm{p})$)

б) $U \leftarrow WY\ mod\ p$ з $0 \leq U < 2p$

в) $X' \leftarrow X + U$

г) $Y' \leftarrow X - U + 2p$

2.3 Припущення про складність алгоритму

Захищеність qTESLA ґрунтується на стійкості проблеми $R-LWE$. В наступне визначення ми використовуємо A^O для позначення того, що A має доступ до оракула O.

Визначення 1 ($R-LWE_{n,k,q,\chi}$). Нехай $n, q > 0$ - цілі числа, χ – розподіл над $R, s \leftarrow \chi$. Визначимо через $D_{s,\chi}$ розподіл $R-LWE$, виводиться за формулою 2.3

$$(a, \langle a, s\rangle + e) \in R_q \times R_q, \text{де } a \leftarrow U(R_q), e \leftarrow \chi \tag{2.3}$$

Дано k кортежі $(a_1, t_1), \ldots, (a_k, t_k)$, вирішальна $R-LWE$ задача $R-LWE_{n,k,q,\chi}$ полягає в тому, щоб розрізнити $(a_i, t_i) \leftarrow U(R_q \times R_q)$ або $(a_i, t_i) \leftarrow D_{s,\chi}$ для всіх i. Перевага $R-LWE$ визначається за формулою 2.4

$$Adv_{n,k,q,\chi}^{R-LWE}(A) = \left|P_r\left[A^{D_{x,s}(\cdot)} = 1\right] - P_r\left[A^{U(R_q \times R_q)(\cdot)} = 1\right]\right| \tag{2.4}$$

Наведене визначення відповідає нормальній формі $R-LWE$ [20], в якій поліноми таємний та помилки дотримуються однакового розподілу χ. У qTESLA χ створюється з D_σ.

2.4 Опис параметрів qTESLA

Системні параметри qTESLA та їх відповідні межі підсумовані «у таблиці 2.1». Параметр λ визначається як параметр захисту, тобто цільовий біт безпеки заданої інстанції. При стандартному налаштуванні R-LWE у нас є $R_q = Z_q[x] / \langle x^n + 1\rangle$, де розмірність n - потужність двох, тобто $n = 2^l$ для $l \in N$.

Параметр $k \in Z > 0$ - це кількість кільцевих знань з помилковими зразками, використані даною інстанцією. Залежно від конкретної функції, параметр k визначає вхідні та/або вихідні довжини хеш-бази та псевдовипадкові функції. Цей параметр визначається як більший або рівний, рівню безпеки λ. Це відповідає використанню хешу у $Fiat - Shamir$ схема підпису стилів, така як qTESLA, для якої релевантний опір попереднього зображення тоді як опір зіткнення набагато менше. Відповідно, ми вважаємо розмір хеша таким, якого має бути достатньо, щоб протистояти передвиборним атакам. Параметр $b_{GenA} \in Z > 0$ представляє кількість запитуваних блоків у першому виклику SHAKE128, під час генерації публічних поліномів $a_1, \ldots, a_k$. Значення b_{GenA} вибираються експериментально, таким чином, щоб вони максимізували продуктивність на цільовій платформі Intel.

Пов'язані параметри та ймовірність прийняття. Значення LS і LE використовуються для обмеження коефіцієнтів поліномів таємниці та помилок у функції оцінки $checkS$ і $checkE$ відповідно. Обмеження розмірів, поліноми обмежують розмір простору ключів; відповідно ми компенсуємо втрату безпеки шляхом вибору більшої жорсткості біт. Обидві межі, L_S і L_E (і, отже, S і E^4), впливають на ймовірність відхилення протягом генерація підписів наступним чином. Якщо збільшувати значення L_S і L_E, то ймовірність прийняття ключа під час генерації, що називається δ_{keygen}, збільшується, при цьому ймовірності прийняття z і w під час генерації підписів, які називаються δ_z та δ_w відповідно, зменшуються. Ми визначаємо гарний компроміс між двома ймовірностями прийняття, під час генерації ключа та експериментального підписання. Для цього ми почнемо з вибору $L_S = \eta_S \cdot h \cdot \sigma$ (відповідно, $L_E = \eta_E \cdot h \cdot \sigma$) з $\eta_S = \eta_E = 2{,}8$ і обчислити відповідні значення параметрів B, d і q (які вибираються, як пояснено далі). Потім ми ретельно налаштовуємо їх параметри шляхом випробування різних значень для η_S та η_E в діапазоні $[2.0, \ldots, 3.0]$ поки ми не знайдемо гарний компроміс між різними ймовірностями і, отже, тривалість виконання. Параметр B визначає інтервал випадкового многочлена y, і він визначається параметрами M і S як представлено у формулі 2.5.

$$(\frac{2B-2S+1}{2B+1})^{k\cdot n} \geq M \Leftrightarrow B \geq \frac{\sqrt[k\cdot n]{M} + 2S - 1}{2\left(1-\sqrt[k\cdot n]{M}\right)} \quad (2.5)$$

Де M - значення нашого вибору. Після вибору B ми вибираємо значення d що визначає функції округлення $[\cdot]M$ і $[\cdot]L$ має бути більше log_2(B) і такий, що ймовірність прийняття перевірки $\|[w]L\|_\infty \geq 2^{d-1} - E$ також, M визначається як нижньо-обмежена. Ця перевірка визначає прийняття ймовірність δ_w під час генерації підписів. Ймовірність прийняття z, а саме δ_z, пов'язана зі значенням M.

Модуль q. Цей параметр вибрано для виконання декількох меж та припущень, мотивованих вимогами ефективності та зниженням безпеки qTESLA. Щоб дозволити використання швидкого множення полінома за допомогою NTT, q має бути простим цілим числом, таким, що $q\ mod\ 2n = 1$. Більше того, ми вибираємо $q > 2B$. Щоб вибрати параметри відповідно до зниження рівня безпеки, спочатку зручно спростити нашу заяву про безпеку. Для цього ми стверджуємо, що $q^{nk} \geq |\Delta S| \cdot |\Delta L| \cdot |\Delta H|$. Тоді має виконуватися рівняння представлене у формі 2.6.

$$\frac{2^{3\lambda+nkd+2}q_s^3(q_s+q_h)^2}{q^{nk}} \leq 2^{-\lambda} \Leftrightarrow q \geq (2^{4\lambda+nkd+2}q_s^3(q_s+q_h)^2)^{\frac{1}{nk}} \quad (2.6)$$

Класичні запити до знаку оракул мають $q_s = min\{2^{64}, 2^{\frac{\lambda}{2}}\}$ для всіх наших наборів параметрів. Крім того, ми обираємо кількість запитів хеш-функції $q_h = min\{2^{128}, 2^{\lambda}\}$.

Розміри ключів та підписів Теоретичні розряди підписів і відкриті ключі задаються $k+n\cdot(\lceil log_2(B-S)\rceil+1)$ і $k\cdot n\cdot(\lceil log_2(q)\rceil e)+k$, відповідно. Щоб визначити розмір секретних ключів, спочатку визначимо t як кількість β-бітових записів таблиць CDT, дискретних гауссових пробовідбірників, що відповідає максимальному значенню, яке може бути вибране для, генерації коефіцієнтів таємних многочленів s. Потім випливає, що теоретичний розмір секретного ключа задається як $n(k+1)(\lceil log_2(t-1)\rceil+1)+2k+320$ бітів.

3 ЕКСПЕРИМЕНТАЛЬНІ ДОСЛІДЖЕННЯ КВАНТОВОГО АЛГОРИТМУ ЦИФРОВОГО ПІДПИСУ QTESLA

3.1 Склад ключів

Особистий ключ:

seeds [0] – випадковий рядок довжиною bytes_of_seed =32 байтів;

seeds [1] - випадковий рядок довжиною bytes_of_seed =32 байтів;

s – поліном з нормальним розподілом ключів, значення коефіцієнтів змінюється в діапазоні [-2 10 .. 2 10]

e – поліноми (кількість поліномів дорівнює K), поліноми з нормальним розподілом ключів, значення коефіцієнтів змінюється в діапазоні [-2 10 .. 2 10]

Відкритий ключ:

а) seeds [0] – випадковий рядок довжиною bytes_of_seed =32 байтів;

б) Поліноми t i = a i * s + e i (всього K поліномів, K = 1).

3.2 Функція генерації ключів (qtesla_keypair)

Вихід.

pk - відкритий ключ (масив октетів)

sk - секретний ключ (масив октетів)

1 Генерація випадкового компонента rand довжиною RANDOMBYTES

2 Кількість Counts = 1 + K + K (seed’s для s, e (e 0 , e 1 , …, e PARAM_K-1), a (a 0 , a 1 ,…, a PARAM_K-1)

3 Генерація Counts напів-випадкових послідовностей довжиною 32 байта кожна, позначимо їх

ext [0], s

ext [1], e

ext [2], a

ext [3], y

4 s:= qtesla_gauss_poly () Генерація поліному s

5 success:= qtesla_check_norma (s)

6 if (success = ERROR) goto 4

7 e:= qtesla_gauss_poly () Генерація поліному e

8 success := qtesla_check_norma (e)

9 if success = ERROR goto 7

10 a:= qtesla_gen_a (ext [2]);

18 t:= a * s + e

19 перетворення секретного ключа в масив байтів sk:=

qtesla_to_string_sk(s, e, ext [2] || ext [3]);

20 перетворення відкритого ключа в масив байтів pk :=

qtesla_to_string_pk(t, ext [2]);

3.3 Формування цифрового підпису (функція qtesla_sign)

Цифровий підпис складається з:

Поліному h, коефіцієнти якого задаються по модулю 3, тобто мають значення -1, 0, 1, поліному z з коефіцієнтами по модулю q

Вхід:

а) m - повідомлення (масив байтів)

б) mlen - довжина повідомлення

в) sk - секретний ключ, масив байтів

Вихід:

а) Цифровий підпис signature довжиною signature_len

б) Алгоритм.

1 Перетворення масива байтів в секретний ключ

seeds[0], seeds[1], s, e := qtesla_decode_sk (sk);

2 input [1]:= випадковий рядок довжиною 32 байти

3 input [0]:= seeds[1]

4 знаходимо хеш для вхідного повідомлення

input [2, 3]:= qtesla_hash64 (m, mlen)

5 Відновлення поліному a:= qtesla_gen_a (seeds [0]);

6 Генерація поліному y:= qtesla_gen_y (); R/q поліном, коефіцієнти в діапазоні [-PARAM_B.. PARAM_B]

7 v = y * a

8 c_ := qtesla_hash_H (v, rand_input [2, 3]) Рядок байтів для поліному [0,-1,1]

9 c := qtesla_gen_c(c_); R/3

10 z:= y + s * c

11 success := qtesla_test_z (z) Перевірка норми z

12 if success = ERROR go to 6

13 v:= v – e * c

14 success = qtesla_test_z (v)

15 if success = ERROR go to 6

16 Перетворення цифрового підпису в рядок байтів

sm, smlen:= qtesla_encode_sig (c, z)

3.4 Перевірка цифрового підпису (функція qtesla_verify)

Вхід:

а) m - повідомлення (масив байтів)

б) mlen - довжина повідомлення

в) sm – цифровий підпис завдовжки smlen (рядок октетів)

г) pk – відкритий ключ (рядок октетів)

Вихід:

а) Success = {OK, ERROR}

1 Success:=ERROR;

2 Перетворення цифрового підпису в поліноми

c_, z:= qtesla_decode_sig (sm, smlen)

3 Success := qtesla_test_z (z)

4 if Success ≠ ERROR then

5 Перетворення масиву байтів для відкритого ключа в відкритий ключ

pk_t, seed := qtesla_decode_pk (pk);

6 a:= qtesla_gen_a (seed);

7 c := qtesla_gen_c(c_); R/3

8 w := a * z - t * c

9 hm = qtesla_hash64 (m, mlen);

10 c’ = qtesla_hash_H(w, hm);

11 if c = c’ then

13 Success = OK

14 end if

15 end if

3.5 Результати тестування продуктивності

Для оцінки ефективності наданих реалізацій я провів тестування продуктивності на комп’ютері з такими характеристиками: процесор Intel(R) Core i7-10510U CPU 2.30 ГГц, під управлінням Linux 5.6.8. Для пакування я використав gcc версії 10.1 з командою gcc -03 -march=native -fomit-frame-pointer.

а) Схема: qTESLA;

б) Генерація ключів: 1,585.0;

в) Підпис: 470.1;

г) Перевірка підпису: 97.7;

д) Підпис з перевіркою, сумарно: 567.8.

Продуктивність (у тисячах циклів) реалізації qTESLA. Підпис здійснюється на повідомленні розміром - 59 байт.

Результати приведені вище, показують високу ефективність даного алгоритму. Час роботи платформи Intel (приблизно) – для підпису:167.0 мс, для перевірки підпису: 254.7 мс, і сумарний час: 426.1 мс. Це демонструє, що швидкість qTESLA є практичною для багатьох застосувань. Маю підкреслити, що ці результати одержані лише із загальною реалізацією на C. Дану реалізацію можна оптимізувати застосуванням векторних інструкцій, щоб отримати подальші покращення продуктивності.

ВИСНОВКИ

У ході виконання науково-дослідної практики було досліджено проблему захисту від квантових комп’ютерів. Були розглянуті алгоритми захисту такі як: qTESLA, Falcon, Dilithium та SPHINCS+. Також були розглянуті особливості квантових алгоритмів, та проблеми квантових обчислень на яких базується постквантовий захист.

Були проведені детальні аналізи алгоритму цифрового підпису - qTESLA, його переваги та недоліки перед конкурентними алгоритмами, докладно розібрали схему підпису даного алгоритму та зробили припущення щодо його складності.

Реалізували алгоритм qTESLA за допомогою мови програмування C, провели тестування продуктивності та привели шлях для ймовірного майбутнього розвитку даного алгоритму.

Підведемо підсумки. Що до захищеності даної схеми підпису: qTESLA, створена з надійно захищеними параметрами. Проблема R-LWE забезпечує високий рівень безпеки даної схеми підпису від атак за допомогою квантового комп’ютера так як і від звичайних атак.

Простота реалізації та масштабованість. qTESLA має дуже компактну структуру, що складається з кількох простих у виконанні функцій. Гауссовий пробовідбір, мабуть, найбільше складна функція в qTESLA, необхідна лише під час генерації ключів.

Висока швидкість і компактність ключів. Підписи qTESLA відносно компактні, що робить їх ідеальними для додатків, у яких розмір підпису надається пріоритету перед розміром відкритого ключа. Відзначимо, що комбінований розмір підпису та публічного ключа досить конкурентноспроможний з іншими ефективними схемами на решітках.

.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

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

2. Boneh, D., Dagdelen, O., Fischlin, M., Lehmann, A., Schaffner, C., ¨ Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (Dec 2011).

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

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

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

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

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

8. Symmetric Key Encryption - why, where and how it's used in banking, URL: https://www.cryptomathic.com/news-events/blog/symmetric-key-encryption-why-where-and-how-its-used-in-banking.

9. FAQ обзорная информация по емаил сертификатам s/mime, URL: https://proverkassl.com/smime_faq.html.

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

11. Алгоритм Шора, URL: https://logic.pdmi.ras.ru/~yura/modern/10modernnote.pdf.

12. Характеристики квантовых компьютеров, URL: https://habr.com/ru/post/458450/.

13. Квантовый алгоритм Гровера, URL: https://sohabr.net/habr/post/204460/.

14. Метод оцінки стійкості алгоритмів обміну ключами в алгоритмах постквантової криптографії SIDH і CSIDH, URL: https://ela.kpi.ua/bitstream/123456789/31308/1/Vlasenko_bakalavr.pdf.

15. Pornin, T.: New efficient, constant-time implementations of Falcon. URL: https://falcon-sign.info/falcon-impl-20190918.pdf (2019), accessed: 2019-10-11.

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

17. Blake-Wilson, S., Menezes, A.: Unknown key-share attacks on the station-to-station (STS) protocol. In: Imai, H., Zheng, Y. (eds.) Public Key Cryptography (PKC'99). Lecture Notes in Computer Science, vol. 1560, pp. 154–170. Springer (1999) [17] Boneh, D., Dagdelen, O., Fischlin, M., Lehmann, A., Schaffner, C., ¨Zhandry, M.: Random oracles in a quantum world. In: Lee, D.H., Wang, X. (eds.) ASIACRYPT 2011. LNCS, vol. 7073, pp. 41–69. Springer, Heidelberg (Dec 2011).

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

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

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

21. Ivan Gorbenko, Yurii Gorbenko, Vladyslav Tymchenko, Olena Kachko TESTING THE SPEED OF MODERN STREAM CIPHERS, URL: https://journals.indexcopernicus.com/api/file/viewByFileId/440351.pdf.

22. Reference implementation of the Kupyna hash function (DSTU 7564:2014), URL: https://github.com/Roman-Oliynykov/Kupyna-reference.

23. Implementation of Strumok stream cipher, URL: https://github.com/outspace/dstu8845.

24. Горбенко І.Д., Качко О.Г., Єсіна М.В, Пономар В.А. Стан та проблемні питання розроблення та впровадження перспективного стандарту цифрового підпису. «КОМП'ЮТЕРНЕ МОДЕЛЮВАННЯ В НАУКОЄМНИХ ТЕХНОЛОГІЯХ» (КМНТ-2020), Харків-2020.

ДОДАТОК А

Вихідний код програми

```
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include "params.h"
#include "poly.h"
#include "pack.h"
#include "rng.h"
#include "hash.h"
#include "gen_polys.h"

void qtesla_hash_H(unsigned char* c_bin, poly *v, const unsigned char* rand)
{
    unsigned char t[PARAM_N * PARAM_K + HM_BYTES];
    int32_t cL, i, j, ii = 0;
     Dstu8845Ctx*pctx = dstu8845_alloc();
    //1  i: = 0
    for (j = 0; j < PARAM_K; j++)
    for (i = 0; i < PARAM_N; i++)
    {
        /*
            2  if v[i] > PARAM_Q/2 then
            3       v[i] := v[i] - PARAM_Q
            4  end if
        */
        if (v[j][i] > PARAM_Q / 2)
            v[j][i] -= PARAM_Q;
        //5  cL = v[i] mod(1 << PARAM_D);
        cL = v[j][i] & ((1 << PARAM_D) - 1);
        // 6  if cL > 2 (d-1) then cL -= 2d
        if (cL > (1 << (PARAM_D - 1)))
            cL -= (1 << PARAM_D);
        // 7 t[i] := ((v[i] - cL) / 2PARAM_D) mod 256;
        t[ii++] = ((v[j][i] - cL) >> PARAM_D) & 0xFF;
```

```
        // 8 i : = i + 1
        // 9 if i < PARAM_N then goto step 2

    }
    // 10 t:= t || rand
    memcpy(t + PARAM_N * PARAM_K, rand, HM_BYTES);
    //11 seedbytes_init(λ, ctx, t, PARAM_N + HM_BYTES)
    seedbytes_init(LAMBDA, pctx, t, PARAM_N * PARAM_K + HM_BYTES);
    //12 randombytes(ctx, c, CRYPTO_C_BYTES);   Додатковий буфер
    randombytes(pctx, c_bin, CRYPTO_C_BYTES);
     dstu8845_free(pctx);
}

static int qtesla_test_correctness(poly v)
{
  int32_t mask, left, val;
  uint32_t t0, t1;
  for (int i=0; i<PARAM_N; i++) {
    mask = (PARAM_Q/2 - v[i]) >> (32-1);
    val = ((v[i]-PARAM_Q) & mask) | (v[i] & ~mask);
    t0 = (uint32_t)(~(Abs(val) - (PARAM_Q/2 - PARAM_REJECTION))) >> (32-1);
    left = val;
    val = (val + (1<<(PARAM_D-1))-1) >> PARAM_D;
    val = left - (val << PARAM_D);

    t1 = (uint32_t)(~(Abs(val) - ((1<<(PARAM_D-1))-PARAM_REJECTION))) >>
(32-1);
    if ((t0 | t1) == 1)
      return 1;
  }
  return 0;
}

static int qtesla_check_z(poly z)
{
  for (int i=0; i<PARAM_N; i++) {
    if (z[i] < -(PARAM_B-PARAM_U) || z[i] > (PARAM_B-PARAM_U))
      return 1;
```

```
  }
  return 0;
}

int qtesla_sign(Dstu8845Ctx* ctx, unsigned char *sm, unsigned long long
*smlen, const unsigned char *m, unsigned long long mlen, const unsigned char*
sk)
{
  unsigned char c[CRYPTO_C_BYTES], randomness[CRYPTO_SEEDBYTES],
randomness_input[CRYPTO_RANDOMBYTES+CRYPTO_SEEDBYTES+HM_BYTES],
seeds[2*CRYPTO_SEEDBYTES];
  uint32_t pos_list[PARAM_H];
  int16_t sign_list[PARAM_H], s[PARAM_N], e[PARAM_K][PARAM_N];
  poly y, v [PARAM_K], Sc, Ec, z, a [PARAM_K];
  int res = 0, j;
  //int nonce = 0;  // Initialize domain separator for sampling y
  Dstu8845Ctx* pctx0 = dstu8845_alloc();
  qtesla_decode_sk(seeds, s, e, sk);
  randombytes(ctx, randomness_input+CRYPTO_RANDOMBYTES, CRYPTO_SEEDBYTES);
  memcpy(randomness_input, &seeds[CRYPTO_SEEDBYTES], CRYPTO_SEEDBYTES);
  hash64(randomness_input + CRYPTO_RANDOMBYTES + CRYPTO_SEEDBYTES, m, mlen);
  hash32(randomness, randomness_input, CRYPTO_RANDOMBYTES + CRYPTO_SEEDBYTES
+ HM_BYTES);
  // ctx0
  seedbytes_init(LAMBDA, pctx0, seeds, CRYPTO_SEEDBYTES);
  qtesla_gen_a(pctx0, a/*, seeds*/);
  seedbytes_init(LAMBDA, pctx0, randomness, CRYPTO_SEEDBYTES);
  while (1) {
      qtesla_gen_y(pctx0, y/*, ++nonce*/);
      for (j = 0; j < PARAM_K; j++)
        poly_mul(v[j], a[j], y);
      qtesla_hash_H(c,  v,  randomness_input  +  CRYPTO_RANDOMBYTES  +
CRYPTO_SEEDBYTES);
      qtesla_gen_c(pos_list, sign_list, c);
      sparse_mul16(Sc, s, pos_list, sign_list);
      poly_add(z, y, Sc);
      if (qtesla_check_z(z) != 0) {
          continue;
```

```
      }
      for (j = 0; j < PARAM_K; j++)
      {
        sparse_mul16(Ec, e[j], pos_list, sign_list);
        poly_sub_correct(v[j], v[j], Ec);
        if (qtesla_test_correctness(v[j]) != 0) {
            break;
        }
      }
       if (j == PARAM_K)
            break;
  }
   for (unsigned long long i = 0; i < mlen; i++)
      sm[CRYPTO_BYTES+i] = m[i];
   *smlen = CRYPTO_BYTES + mlen;
   qtesla_encode_sig(sm, c, z);
     dstu8845_free(pctx0);
  return 0;
}

int qtesla_verify(unsigned char *m, unsigned long long *mlen, const unsigned
char *sm, unsigned long long smlen, const unsigned char *pk)
{
  unsigned       char       c[CRYPTO_C_BYTES],       c_sig[CRYPTO_C_BYTES],
seed[CRYPTO_SEEDBYTES], hm[HM_BYTES];
  uint32_t pos_list[PARAM_H];
  int16_t sign_list[PARAM_H];
  int32_t pk_t[PARAM_K][PARAM_N];
  poly w [PARAM_K], z, a [PARAM_K], Tc;
  int32_t j;
  if (smlen < CRYPTO_BYTES) return -1;
  qtesla_decode_sig(c, z, sm);
  if (qtesla_check_z(z) != 0) return -2;                  // Check norm of z
  qtesla_decode_pk(pk_t, seed, pk);

  Dstu8845Ctx *pctx0 = dstu8845_alloc();
  seedbytes_init(LAMBDA, pctx0, seed, CRYPTO_SEEDBYTES);
  qtesla_gen_a(pctx0, a/*, seed*/);
```

```
  qtesla_gen_c(pos_list, sign_list, c);
  for (j = 0; j < PARAM_K; j++)
  {
      sparse_mul32(Tc, pk_t[j], pos_list, sign_list);
      poly_mul(w[j], a[j], z);
      poly_sub_reduce(w[j], w[j], Tc);                    // Compute w = az
- tc
  }
  hash64(hm, sm + CRYPTO_BYTES, smlen - CRYPTO_BYTES);
  qtesla_hash_H(c_sig, w, hm);
  if (memcmp(c, c_sig, CRYPTO_C_BYTES)) return -3;
  *mlen = smlen-CRYPTO_BYTES;
  for (unsigned long long i = 0; i < *mlen; i++)
    m[i] = sm[CRYPTO_BYTES+i];
  dstu8845_free(pctx0);
  return 0;
}
```

Фрагмент коду 1 - Підпис та перевірка підпису

```
static const int64_t cdt_v[CDT_ROWS*CDT_COLS] = {
      0x0000000000000000LL, // 0
      0x023A1B3F94933202LL, // 1  160470696614638082
      0x06AD3C4C19410B24LL, // 2
      0x0B1D1E95803CBB73LL, // 3
      0x0F879D85E7AB7F6FLL, // 4
      0x13EA9C5C52732915LL, // 5
      0x18440933FFD2011BLL, // 6
      0x1C91DFF191E15D07LL, // 7
      0x20D22D0F2017900DLL, // 8
      0x25031040C1E626EFLL, // 9
      0x2922BEEBA163019DLL, // 10
      0x2D2F866A3C5122D3LL, // 11
      0x3127CE192059EF64LL, // 12
      0x350A1928231CB01ALL, // 13
      0x38D5082CD4FCC414LL, // 14
      0x3C875A73B33ADA6BLL, // 15
```

```
0x401FEF0E67CD47D3LL, // 16
0x439DC59E3077B59CLL, // 17
0x46FFFEDA4FC0A316LL, // 18
0x4A45DCD32E9CAA91LL, // 19
0x4D6EC2F3922E5C24LL, // 20
0x507A35C1FB354670LL, // 21
0x5367DA64EA5F1C63LL, // 22
0x563775ED5B93E26ELL, // 23
0x58E8EC6B50CB95F8LL, // 24
0x5B7C3FD0B999197DLL, // 25
0x5DF18EA7664D810ELL, // 26
0x6049129F03B5CD6DLL, // 27
0x62831EF856A48427LL, // 28
0x64A01ED314BA206FLL, // 29
0x66A09363CA89DAA3LL, // 30
0x688512173EF213F5LL, // 31
0x6A4E42A8B137E138LL, // 32
0x6BFCDD302C5B888ALL, // 33
0x6D91A82DF797EAB8LL, // 34
0x6F0D7697EBA6A51DLL, // 35
0x707125ED27F05CF1LL, // 36
0x71BD9C544C184D8DLL, // 37
0x72F3C6C7FB380322LL, // 38
0x74149755088E5CC6LL, // 39
0x7521036D434271D4LL, // 40
0x761A02516A02B0CELL, // 41
0x77008B9461817A43LL, // 42
0x77D595B95BC6A0FELL, // 43
0x789A14EE338BB727LL, // 44
0x794EF9E2D7C53213LL, // 45
0x79F530BE414FE24DLL, // 46
0x7A8DA03110886732LL, // 47
0x7B1928A59B3AA79ELL, // 48
0x7B98A38CE58D06AELL, // 49
0x7C0CE2C7BAD3164ALL, // 50
0x7C76B02ADDE64EF2LL, // 51
0x7CD6CD1D13EE98F2LL, // 52
0x7D2DF24DA06E2473LL, // 53
```

```
0x7D7CCF81A5CD98B9LL, // 54
0x7DC40B76C24FB5D4LL, // 55
0x7E0443D92DE22661LL, // 56
0x7E3E0D4B91401720LL, // 57
0x7E71F37EC9C1DE8DLL, // 58
0x7EA07957CE6B9051LL, // 59
0x7ECA1921F1AF6404LL, // 60
0x7EEF44CBC73DA35BLL, // 61
0x7F10662D0574233DLL, // 62
0x7F2DDF53CDDCD427LL, // 63
0x7F480AD7DF028A76LL, // 64
0x7F5F3C324B0F66B2LL, // 65
0x7F73C018698C18A7LL, // 66
0x7F85DCD8D69F8939LL, // 67
0x7F95D2B96ED3DA10LL, // 68
0x7FA3DC55532D71BBLL, // 69
0x7FB02EFA1DDDC61ELL, // 70
0x7FBAFB038BAE76E4LL, // 71
0x7FC46C34F918B3E3LL, // 72
0x7FCCAA102B95464CLL, // 73
0x7FD3D828F7D49092LL, // 74
0x7FDA16756C11CF83LL, // 75
0x7FDF819A3A7BFE69LL, // 76
0x7FE4333332A5FEBDLL, // 77
0x7FE84217AA0DE2B3LL, // 78
0x7FEBC29AC3100A8BLL, // 79
0x7FEEC6C78F0D514ELL, // 80
0x7FF15E9914396F2ALL, // 81
0x7FF3982E4982FB97LL, // 82
0x7FF57FFA236862D1LL, // 83
0x7FF720EFD36F4850LL, // 84
0x7FF884AB61732BC7LL, // 85
0x7FF9B396CA3B383CLL, // 86
0x7FFAB50BD1DD3633LL, // 87
0x7FFB8F72BA84114BLL, // 88
0x7FFC485E115A3388LL, // 89
0x7FFCE4A3C3B92B98LL, // 90
0x7FFD6873AE755E4ALL, // 91
```

```
0x7FFDD76BD840FDA1LL, // 92
0x7FFE34AA86CE6870LL, // 93
0x7FFE82DE5CA6A885LL, // 94
0x7FFEC454ABAA26DFLL, // 95
0x7FFEFB0625FADB89LL, // 96
0x7FFF28A214B1160FLL, // 97
0x7FFF4E983945429DLL, // 98
0x7FFF6E217C168A6ALL, // 99
0x7FFF884787F2B986LL, // 100
0x7FFF9DEB70088602LL, // 101
0x7FFFAFCB7B419E48LL, // 102
0x7FFFBE882DABB8F8LL, // 103
0x7FFFCAA8A65BDA07LL, // 104
0x7FFFD49E66188754LL, // 105
0x7FFFDCC891191605LL, // 106
0x7FFFE376BC4B0583LL, // 107
0x7FFFE8EB54D33209LL, // 108
0x7FFFED5DAEE78F4ELL, // 109
0x7FFFF0FBC7A6933DLL, // 110
0x7FFFF3EBC43A9213LL, // 111
0x7FFFF64D375FC4CCLL, // 112
0x7FFFF83A354A0431LL, // 113
0x7FFFF9C83CE9BB0DLL, // 114
0x7FFFFB08FCAC61A6LL, // 115
0x7FFFFC0AF80A1A6FLL, // 116
0x7FFFFCDA127DDE76LL, // 117
0x7FFFFD8003E62E56LL, // 118
0x7FFFFE04B9BF9C5BLL, // 119
0x7FFFFE6EA82EF9BDLL, // 120
0x7FFFFEC30D64CD46LL, // 121
0x7FFFFF0629856684LL, // 122
0x7FFFFF3B6CEEE3F1LL, // 123
0x7FFFFF659E6F7BA6LL, // 124
0x7FFFFF86FAC1036ALL, // 125
0x7FFFFFA14E69EDE9LL, // 126
0x7FFFFFB60AF6ACB7LL, // 127
0x7FFFFFC65857AECFLL, // 128
0x7FFFFFD3230F314FLL, // 129
```

```
0x7FFFFFDD27BE0A17LL, // 130
0x7FFFFFE4FC86CDFFLL, // 131
0x7FFFFFEB18AA9E4CLL, // 132
0x7FFFFFEFDAB1FD73LL, // 133
0x7FFFFFF38D65D499LL, // 134
0x7FFFFFF66BD0EB8CLL, // 135
0x7FFFFFF8A4782371LL, // 136
0x7FFFFFFA5BEF7C27LL, // 137
0x7FFFFFFBAEEB0B4CLL, // 138
0x7FFFFFFCB3E55903LL, // 139
0x7FFFFFFD7C6FE192LL, // 140
0x7FFFFFFE163E99E3LL, // 141
0x7FFFFFFE8BFC2558LL, // 142
0x7FFFFFFEE5F1CE80LL, // 143
0x7FFFFFFF2A8C31FDLL, // 144
0x7FFFFFFF5EC3CD18LL, // 145
0x7FFFFFFF866F376BLL, // 146
0x7FFFFFFFA483A906LL, // 147
0x7FFFFFFFBB4780C4LL, // 148
0x7FFFFFFFCC79BEB2LL, // 149
0x7FFFFFFFD970CBE1LL, // 150
0x7FFFFFFFE3326D21LL, // 151
0x7FFFFFFFEA865AB8LL, // 152
0x7FFFFFFFF004A7C8LL, // 153
0x7FFFFFFFF420E4F9LL, // 154
0x7FFFFFFFF732B791LL, // 155
0x7FFFFFFFF97C764FLL, // 156
0x7FFFFFFFFB303DDDLL, // 157
0x7FFFFFFFFC73D5A3LL, // 158
0x7FFFFFFFFD63AA57LL, // 159
0x7FFFFFFFFE15140DLL, // 160
0x7FFFFFFFFE981196LL, // 161
0x7FFFFFFFFEF89992LL, // 162
0x7FFFFFFFFF3F9A0CLL, // 163
0x7FFFFFFFFF73BA0BLL, // 164
0x7FFFFFFFFF99EBBBLL, // 165
0x7FFFFFFFFFB5DAA0LL, // 166
0x7FFFFFFFFFCA3E7BLL, // 167
```

```
0x7FFFFFFFFD91985LL, // 168
0x7FFFFFFFFE3E70ALL, // 169
0x7FFFFFFFFEBBE45LL, // 170
0x7FFFFFFFFF16C5CLL, // 171
0x7FFFFFFFFF587BELL, // 172
0x7FFFFFFFFF87E7FLL, // 173
0x7FFFFFFFFFAA108LL, // 174
0x7FFFFFFFFFC29F5LL, // 175
0x7FFFFFFFFFD43E8LL, // 176
0x7FFFFFFFFFE0DD7LL, // 177
0x7FFFFFFFFFE9E31LL, // 178
0x7FFFFFFFFFF0530LL, // 179
0x7FFFFFFFFFF4E88LL, // 180
0x7FFFFFFFFFF82AALL, // 181
0x7FFFFFFFFFFA7A6LL, // 182
0x7FFFFFFFFFFC1D6LL, // 183
0x7FFFFFFFFFFD458LL, // 184
0x7FFFFFFFFFFE166LL, // 185
0x7FFFFFFFFFFEA97LL, // 186
0x7FFFFFFFFFFF10CLL, // 187
0x7FFFFFFFFFFF594LL, // 188
0x7FFFFFFFFFFF8C0LL, // 189
0x7FFFFFFFFFFFAF7LL, // 190
0x7FFFFFFFFFFFC83LL, // 191
0x7FFFFFFFFFFFD96LL, // 192
0x7FFFFFFFFFFFE56LL, // 193
0x7FFFFFFFFFFFEDALL, // 194
0x7FFFFFFFFFFFF36LL, // 195
0x7FFFFFFFFFFFF75LL, // 196
0x7FFFFFFFFFFFFA1LL, // 197
0x7FFFFFFFFFFFFBFLL, // 198
0x7FFFFFFFFFFFFD4LL, // 199
0x7FFFFFFFFFFFFE2LL, // 200
0x7FFFFFFFFFFFFECLL, // 201
0x7FFFFFFFFFFFFF2LL, // 202
0x7FFFFFFFFFFFFF7LL, // 203
0x7FFFFFFFFFFFFFALL, // 204
0x7FFFFFFFFFFFFFCLL, // 205
```

```
      0x7FFFFFFFFFFFFFDLL, // 206
      0x7FFFFFFFFFFFFFELL, // 207
      0x7FFFFFFFFFFFFFFLL, // 208
}; // cdt_v
#define DFIELD ((int64_t)(~(uint64_t)0 >> 1))
#define PRODIFF(diff, a_u, a_v, k) { \
    diff = (diff + (a_v[k] & DFIELD) - (a_u[k] & DFIELD)) >> (64-1); \
}
#define PROSWAP(swap, diff, a_u, a_v, k) { \
    swap = (a_u[k] ^ a_v[k]) & diff; a_u[k] ^= swap; a_v[k] ^= swap; \
}
#define PROSWAPG(swap, diff, g_u, g_v) { \
    swap = (g_u ^ g_v) & (int32_t)diff; g_u ^= swap; g_v ^= swap; \
}
#define MINMAX0(swap, diff, a_u, a_v) { \
    PRODIFF(diff, a_u, a_v, 0); \
    PROSWAP(swap, diff, a_u, a_v, 0); \
}
#if CDT_COLS > 1
#define MINMAX1(swap, diff, a_u, a_v) { \
        PRODIFF(diff, a_u, a_v, 1); \
        MINMAX0(swap, diff, a_u, a_v); \
        PROSWAP(swap, diff, a_u, a_v, 1); \
    }
#else
#define MINMAX1(swap, diff, a_u, a_v) MINMAX0(swap, diff, a_u, a_v)
#endif
#if CDT_COLS > 2
#define MINMAX2(swap, diff, a_u, a_v) { \
        PRODIFF(diff, a_u, a_v, 2); \
        MINMAX1(swap, diff, a_u, a_v); \
        PROSWAP(swap, diff, a_u, a_v, 2); \
    }
#else
#define MINMAX2(swap, diff, a_u, a_v) MINMAX1(swap, diff, a_u, a_v)
#endif
#if CDT_COLS > 3
#define MINMAX3(swap, diff, a_u, a_v) { \
```

```
        PRODIFF(diff, a_u, a_v, 3); \
        MINMAX2(swap, diff, a_u, a_v); \
        PROSWAP(swap, diff, a_u, a_v, 3); \
    }
#else
#define MINMAX3(swap, diff, a_u, a_v) MINMAX2(swap, diff, a_u, a_v)
#endif
#if CDT_COLS > 4
#define MINMAX4(swap, diff, a_u, a_v) { \
        PRODIFF(diff, a_u, a_v, 4); \
        MINMAX3(swap, diff, a_u, a_v); \
        PROSWAP(swap, diff, a_u, a_v, 4); \
    }
#else
#define MINMAX4(swap, diff, a_u, a_v) MINMAX3(swap, diff, a_u, a_v)
#endif
#if CDT_COLS <= 5
     // TODO: improve MINIMAX performance:
#define MINIMAX(a_u, a_v, g_u, g_v) { \
        int64_t diff = 0, swapa; int32_t swapg; \
        MINMAX4(swapa, diff, a_u, a_v); \
        PROSWAPG(swapg, diff, g_u, g_v); \
    }
#else
#error "Unsupported precision"
#endif
#define MINMAXG(a_u, a_v) {\
    int32_t diff = ((a_v & 0x7FFFFFFFL) - (a_u & 0x7FFFFFFFL)) >> (RADIX32-
1); \
    int32_t swap = (a_u ^ a_v) & diff; a_u ^= swap; a_v ^= swap; \
}

// Сортування обєднанням
static   void   knuthMergeExchangeKG(int64_t   a[/*n*CDT_COLS*/],   int32_t
g[/*n*/], unsigned int n)
{
     unsigned int t = 1;
     while (t < n - t) {
```

```
            t += t;
      }
      for (unsigned int p = t; p > 0; p >>= 1) {
            int64_t *ap = a + p * CDT_COLS;
            int64_t *a_i = a, *ap_i = ap;
            int32_t *gp = g + p;
            for (unsigned int i = 0; i < (unsigned int)(n - p); i++, a_i +=
CDT_COLS, ap_i += CDT_COLS) {
                  if (!(i & p)) {
                        MINIMAX(a_i, ap_i, g[i], gp[i]);
                  }
            }
            for (unsigned int q = t; q > p; q >>= 1) {
                  int64_t *ap_i = ap, *aq_i = a + q * CDT_COLS;
                  int32_t *gq = g + q;
                  for (unsigned int i = 0; i < (unsigned int)(n - q); i++,
ap_i += CDT_COLS, aq_i += CDT_COLS) {
                        if (!(i & p)) {
                              MINIMAX(ap_i, aq_i, gp[i], gq[i]);
                        }
                  }
            }
      }
}

// Сортування обєднанням
static void knuthMergeExchangeG(int32_t a[/*n*/], size_t n) {
      if (n <= 1) {
            return;
      }
      size_t t = 1;
      while (t < n - t) {
            t += t;
      }
      for (size_t p = t; p > 0; p >>= 1) {
            int32_t *ap = a + p;
            for (size_t i = 0; i < n - p; i++) {
                  if (!(i & p)) {
```

```
                    MINMAXG(a[i], ap[i]);
                }
            }
            for (size_t q = t; q > p; q >>= 1) {
                int32_t *aq = a + q;
                for (size_t i = 0; i < n - q; i++) {
                    if (!(i & p)) {
                        MINMAXG(ap[i], aq[i]);
                    }
                }
            }
        }
}

typedef int64_t         sdigit_t;       // Signed 64-bit digit
#define CHUNK_SIZE 512
// Генерація полінома з коефіцієнтами згідно розподілу Гауса
void  qtesla_gen_gauss_poly/*qtesla_kmxGauss*/(Dstu8845Ctx*  ctx,  int32_t
z[]/*, const unsigned char* seed*/)
{
    sdigit_t sampk[(CHUNK_SIZE + CDT_ROWS) * CDT_COLS];
    int32_t sampg[CHUNK_SIZE + CDT_ROWS];
    randombytes(ctx,   (uint8_t*)sampk,   CHUNK_SIZE   *   CDT_COLS   *
sizeof(sdigit_t));
    memcpy(sampk + CHUNK_SIZE * CDT_COLS, cdt_v, CDT_ROWS * CDT_COLS *
sizeof(sdigit_t));
    for (int32_t i = 0; i < CHUNK_SIZE; i++)
        sampg[i] = i << 16;
    for (int32_t i = 0; i < CDT_ROWS; i++)
        sampg[CHUNK_SIZE + i] = 0xFFFF0000L ^ i;
    knuthMergeExchangeKG(sampk, sampg, CHUNK_SIZE + CDT_ROWS);
    int32_t prev_inx = 0;
    for (int i = 0; i < CHUNK_SIZE + CDT_ROWS; i++) {
        int32_t curr_inx = sampg[i] & 0xFFFFL;
        prev_inx ^= (curr_inx ^ prev_inx) & ((prev_inx - curr_inx) >>
(RADIX32 - 1));
        int32_t neg = (int32_t)(sampk[i * CDT_COLS] >> (RADIX - 1));  //
Only the (so far unused) msb of the leading word
```

```
            sampg[i] |= ((neg & -prev_inx) ^ (~neg & prev_inx)) & 0xFFFFL;
      }
      knuthMergeExchangeG(sampg, CHUNK_SIZE + CDT_ROWS);
      for (int i = 0; i < CHUNK_SIZE; i++) {
            z[i] = (sampg[i] << (RADIX32 - 16)) >> (RADIX32 - 16);
      }
}

// Перевірка норми для малих поліномів
static int qtesla_check_ES(poly p, unsigned int bound)
{
      unsigned int i, j, sum = 0, limit = PARAM_N;
      int32_t temp, mask, list[PARAM_N];
      for (j = 0; j < PARAM_N; j++)
            list[j] = Abs(p[j]);
      for (j = 0; j < PARAM_H; j++) {
            for (i = 0; i < limit - 1; i++) {
                  // If list[i+1] > list[i] then exchange contents
                  mask = (list[i + 1] - list[i]) >> (32 - 1);
                  temp = (list[i + 1] & mask) | (list[i] & ~mask);
                  list[i + 1] = (list[i] & mask) | (list[i + 1] & ~mask);
                  list[i] = temp;
            }
            sum += (unsigned int)list[limit - 1];
            limit -= 1;
      }
      if (sum > bound)
            return 1;
      return 0;
}

// Генерація ключів
int qtesla_keypair(Dstu8845Ctx* ctx, unsigned char *pk, unsigned char *sk)
{
      unsigned char randomness[CRYPTO_RANDOMBYTES], randomness_extended[4 *
CRYPTO_SEEDBYTES];
      poly s, e[PARAM_K], a[PARAM_K], t[PARAM_K];
      int i, j;
```

```
Dstu8845Ctx *ctx0 = dstu8845_alloc();
int counts = 1 +  // SEED_S
     1 +     // SEED_E
     1 +     // SEED_A
     1;      // SEED_Y
randombytes(ctx, randomness, CRYPTO_RANDOMBYTES);
seedbytes_init(LAMBDA, ctx0, randomness, CRYPTO_RANDOMBYTES);
randombytes(ctx0, randomness_extended, counts * CRYPTO_SEEDBYTES);
seedbytes_init(LAMBDA,       ctx0,       randomness_extended       /*+
CRYPTO_SEEDBYTES*/, CRYPTO_SEEDBYTES);
do {  // Sample the secret polynomial
     qtesla_gen_gauss_poly(ctx0, s);
} while (qtesla_check_ES(s, PARAM_KEYGEN_BOUND_S) != 0);
uint8_t* prand = randomness_extended + CRYPTO_SEEDBYTES;
seedbytes_init(LAMBDA, ctx0, prand, CRYPTO_SEEDBYTES);
for (j = 0; j < PARAM_K; j++)
{
     for (i = 0; i < PARAM_K; i++)
     {
          do {
               qtesla_gen_gauss_poly(ctx0, e[i]);
          } while (qtesla_check_ES(e[i], PARAM_KEYGEN_BOUND_E) != 0);
     }
}
prand += CRYPTO_SEEDBYTES;
seedbytes_init(LAMBDA, ctx0, prand , CRYPTO_SEEDBYTES);
qtesla_gen_a(ctx0, a);
for (i = 0; i < PARAM_K; i++)
{
     poly_mul(t[i], a[i], s);
     poly_add_correct(t[i], t[i], e[i]);
}
qtesla_encode_sk(sk, s, e, prand);
qtesla_encode_pk(pk, t, prand);
dstu8845_free(ctx);
return 0; }
```

Фрагмент коду 2 – Генерація ключів та передбачення для розподілу Гауса