

Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет _____ Комп'ютерних наук _____

Кафедра _____ Програмної інженерії _____

АТЕСТАЦІЙНА РОБОТА

Пояснювальна записка

рівень вищої освіти – другий (магістерський)

Дослідження постквантових алгоритмів. Цифровий підпис

Виконав: студент 2 курсу, групи ІПЗм-18-1 _____

_____ Ворожка С.О. _____

(прізвище, ініціали)

спеціальності 121- Інженерія програмного забезпечення

(код і повна назва спеціальності)

Освітньо-професійної програми _____

(тип програми)

Програмне забезпечення систем _____

(повна назва освітньої програми)

Керівник _____ проф. Качко О.Г. _____

(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри, проф. _____

З.В.Дудар

Харківський національний університет радіоелектроніки

Факультет Комп'ютерних наук

Кафедра Програмної інженерії

Рівень вищої освіти - другий (магістерський)

Спеціальність 121-Інженерія програмного забезпечення

(код і повна назва)

Тип програми освітньо-професійна програма

Освітня програма Програмне забезпечення систем

ЗАТВЕРДЖУЮ:

Зав. кафедри _____

(підпис)

« ____ » _____ 20 ____ р.

ЗАВДАННЯ

НА АТЕСТАЦІЙНУ РОБОТУ

студентові Ворожку Станіславу Олександровичу

(прізвище, ім'я, по батькові)

1. Тема роботи Дослідження постквантумних алгоритмів. Цифровий підпис затверджена наказом університету від “ ____ ” _____ 20 ____ р № _____
2. Термін подання студентом роботи до екзаменаційної комісії _____
3. Вихідні дані до роботи алгоритм Шора, постквантумні алгоритми, qTESLA, цифровий підпис.
4. Перелік питань, що потрібно опрацювати в роботі мета роботи, аналіз проблемної галузі і постановка задачі, глибокий аналіз роботи алгоритму qTESLA, реалізація алгоритму qTESLA.

5 Консультанти розділів роботи

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Спецчастина	.		

КАЛЕНДАРНИЙ ПЛАН

№	Назва етапів роботи	Терміни виконання етапів роботи	Примітка*
1.	Аналіз предметної галузі	15 лютого 2020р.	
2.	Огляд алгоритмів постквантового шифрування	10 березня 2020р.	
3.	Моделювання алгоритму qTESLA	17 березня 2020р.	
4.	Підготовка пояснювальної записки	23 березня 2020р.	
5.	Спецчастина	13 квітня 2020р.	
6.	Підготовка презентації та доповіді	18 квітня 2020р.	
7.	Попередній захист	12 травня 2020р.	
8.	Нормоконтроль, рецензування	12 травня 2020р.	
9.	Занесення диплома в електронний архів	13 травня 2020р.	
10.	Допуск до захисту у зав. кафедри	13 травня 2020р.	

Дата видачі завдання _____ 2020 р.

Студент _____

(підпис)

Керівник роботи _____ проф. Качко О.Г.

(підпис)

(посада, прізвище, ініціали)

РЕФЕРАТ / ABSTRACT

Пояснювальна записка до атестаційної роботи магістра, 36 с., 2 таблиці, 24 джерел.

АТЕСТАЦІЙНА РОБОТА, ЦИФРОВЕ ШИФРУВАННЯ, ЕЛЕКТРОННИЙ ЦИФРОВИЙ ПІДПИС, АЛГОРИТМ ШОРА, КВАНТОВИЙ КОМП'ЮТЕР, ПОСТКВАНТОВА КРИПТОГРАФІЯ, QTESLA, СХЕМА ПІДПISУ.

Об'єктом дослідження є постквантові алгоритми цифрового підпису, в основі криптостійкості яких є важкі задачі, пов'язані з пошуком найкоротшого вектору в решітках.

Метою роботи є дослідження методів та алгоритмів захисту від квантових комп'ютерів.

У результаті роботи було досліджені та порівняні алгоритми постквантового цифрового підпису, детально проаналізовано та реалізовано алгоритм цифрового підпису qTESLA, визначені швидкісні характеристики цього алгоритму.

ATTESTATION WORK, DIGITAL ENCRYPTION, ELECTRONIC DIGITAL SIGNATURE, SHORA ALGORITHM, QUANTUM COMPUTER, POST-QUANTUM CRYPTOGRAPHY, QTESLA, SIGNATURE SCHEME

The object of the study is the post-quantum digital signature algorithms, which are based on cryptocurrency's difficult tasks associated with finding the shortest vector in the lattice.

The purpose of this work is to investigate methods and algorithms for protection against quantum computers.

As a result, the post-quantum digital signature algorithms were investigated and compared, the qTESLA digital signature algorithm was analyzed and implemented in detail, and the speed characteristics of this algorithm were determined.

ЗМІСТ

Вступ.....	6
1 Аналіз предметної області.....	7
1.1 Як працює цифрове шифрування?	7
1.2 Що таке електронний цифровий підпис?	8
1.3 Чому квантові комп'ютери несуть загрозу шифрування?.....	9
1.4 Чи зможуть квантові комп'ютери зламувати криптографічний захист найближчим часом?	10
1.5 Якою буде постквантова криптографія?.....	11
1.6 Порівняння з аналогами	12
2 Аналіз алгоритму qTESLA	14
2.1 Схема підпису qTESLA	14
2.2 Число-теоретичне перетворення (NTT).....	19
2.2.1 Алгоритм метелика	21
2.2.2 Модульне множення.....	22
2.2.3 Видалення «if» кроків.....	23
2.2.4 Зворотній алгоритм метелика	24
2.3 Припущення про складність алгоритму	25
2.4 Опис параметрів qTESLA.....	25
3 Цифровий підпис qTESLA	28
3.1 Склад ключів	28
3.2 Функція генерації ключів (qtesla_keypair).....	28
3.3 Формування цифрового підпису (функція qtesla_sign).....	29
3.4 Перевірка цифрового підпису (функція qtesla_verify)	31
4 Результати тестування продуктивності	32
Висновки	33
Перелік джерел посилання	34
Додаток А Вихідний код програми	37
Додаток Б Слайди презентації	54

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, наввипередки розробляють нові підходи до криптографії, які зможуть витримати майбутні квантові атаки хакерів.

1.1 Як працює 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 звані 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 Що таке 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 іншим особам (навіть центру сертифікації ключів). Будь-хто може перевірити цифровий підпис, використовуючи тільки відкритий ключ.

1.3 Чому квантові комп'ютери несуть загрозу шифрування?

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 кубітами 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.

1.4 Чи зможуть квантові комп'ютери зламувати криптографічний захист найближчим часом?

Малоймовірно. 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'ютеру знадобиться мільярд кубітів. У більш сучасній 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 кубітами. 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 постквантову криптографію.

1.5 Якою буде 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 ізогена [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, компактным і надійним дизайном.

2 АНАЛІЗ АЛГОРИТМУ 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 опису 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, \dots, a_k	-
Q	модуль	$q = 1 \pmod{2n}, q$ $> 2B, q2^{d+1}q^{nk}$ $\geq \Delta S \cdot \Delta L \cdot \Delta H ,$ $q^{nk} \geq 2^{4\lambda+nkd} 4q_S^3 (q_S + q_h)^2$
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	$d > \log_2(B), d \geq \log_2\left(\frac{2E + 1}{1 - M \frac{1}{nk}}\right)$
H	№ ненульових записів вихідних елементів E_{nc}	$2^h \cdot \binom{n}{h} \geq 2^{2\lambda}$
b_{Gen}	№ блоку запитані SHAKE128 для Gen	$b_{Gen} \in Z > 0$

Для реалізації схеми необхідні наступні функції:

а) Псевдовипадкові функції $PRF_1: \{0, 1\}^k \rightarrow \{0, 1\}^{k, k+3}$, що приймає як вхідний параметр *seed pre-seed*, який є довгим k , і відобразить його на $(k + 3)$ *seed* k біт кожен.

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

в) Псевдовипадкова функція $PRF_2: \{0, 1\}^k \times \{0, 1\}^k \times \{0, 1\}^{320} \rightarrow \{0, 1\}^k$, яке приймає як вхідне значення *seed* та випадкове значення r , кожен k біт довгий, і хеш G повідомлення m , який має розмір 320 біт, і відображає їх у k -бітний *seed rand*.

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

д) Гауссова функція пробо-відбору $GaussSampler: \{0, 1\}^k \times Z \rightarrow R$, який приймає за вхід k -бітове *seed* $\in \{seed_s, seede_{e_1}, \dots, seede_{e_k}\}$ і *nonce counter* $\in Z > 0$, і виводить многочлен в R , відібраний відповідно до дискретного гауссового розподілу D_σ .

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

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

к) Хеш-функція $H: R_q^k \times \{0, 1\}^{320} \times \{0, 1\}^{320} \rightarrow \{0, 1\}^k$. Це функція приймає як входи k многочленів $v_1, \dots, v_k \in R_q$ і перші обчислення $[v_1]_M, \dots, [v_k]_M$. Потім результат хеширують разом з хешем $G(m)$ для заданого повідомлення m і хеша $G(t_1, \dots, t_k)$ до рядка k 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 $i = 1, \dots, k$ 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, \dots, 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, \dots, e_k . У такому випадку ці поліноми повинні виконати перевірку правильності в $checkE$.

Для генерації псевдовипадкових бітових рядків під час дискретного гауссова вибірки відповідне значення від $\{seed_s, seed_{e_1}, \dots, seed_{e_k}\}$ 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 $t_i = a_i s + e_i \bmod q$ при $i = 1, \dots, k$, і секретний ключ sk складається з $s, e_1, \dots, e_k, seed_a$ і $seed_y$, і хеш $g = G(t_1, \dots, t_k)$. Всі $seed$, необхідні під час генерації ключа, генеруються компанією розширення pre-seed за допомогою PRF_1 .

Генерація підписів. Щоб підписати повідомлення m , спочатку многочлен $u \in R_{[B]}$ вибирається рівномірно випадково. З цією метою лічильник, ініціалізований на одиниці, використовується як *nonce*, так і випадковий рядок - *rand*, обчислений як $PRF_2(\text{seed}_y, r, G(m))$ 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 u . 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, \dots, a_k які потім використовуються для обчислення поліномів $v_i = a_i u \bmod \pm q$ для $i = 1, \dots, k$. Після цього хеш-функція обчислює $[v_1]_M, \dots, [v_k]_M$ і хеширує їх разом із дайджестами $G(m)$ і g для отримання c' . Потім це значення детерміновано відображається на генерований псевдовипадковий многочлен $c \in H_{n,h}$, кодований у вигляді двох масивів $\text{pos_list} \in \{0, \dots, n - 1\}^h$ і $\text{sign_list} \in \{-1, 1\}^h$ що представляють положення та ознаки ненульових коефіцієнтів c відповідно. Для того, щоб потенційний підпис $(z \leftarrow sc + u, c')$ 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 u . 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 $\|[w_i]\|_\infty < [q/2] - E$. Якщо перевірка не вдається, то алгоритм відкидає поточну пару (z, c') і повторює всі кроки, що починаються вибіркою u . 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 $a_1, \dots, a_k \in R_q$, а потім обчислює $w_i = a_i z - b_i c \bmod \pm q$ для $i = 1, \dots, k$. Хеш-функція H обчислює $[w_1]_M, \dots, [w_k]_M$ і хеширує їх разом із дайджестами $G(m)$ і $G(t_1, \dots, t_k)$. 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 $i = 1, \dots, k$, $[a_i z - t_i c]_M = [a_i (y + sc) - (a_i s + e_i) c]_M = [a_i y - e_i c]_M = [a_i y]_M$.

Особливості дизайну. Дизайн qTESLA оснащений кількома вбудованими функціями безпеки. По-перше, публічні многочлени a_1, \dots, 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, \dots, 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 k 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, \dots, 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 u при кожній операції підписання, яка робить підписи qTESLA вірогідними. Ймовірнісні підписи, можливо, складніше атакувати за допомогою аналізу бічних каналів. Причому свіжий u запобігає легким u 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 мавуть 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 $q \equiv 1 \pmod{2n}$, нехай $a = (a_0, \dots, a_{n-1}) \in Z_q^n$, і нехай ω буде примітивний n -й корінь в Z_q , що означає, що $\omega^n \equiv 1 \pmod{q}$. Пряме перетворення $\tilde{a} = NTT(a)$ визначається як $\tilde{a}_i = \sum_{j=0}^{n-1} a_j \omega^{ij} \pmod{q}$ для $i = 0, 1, \dots, n-1$. Зворотнє перетворення задається як $b = INTT(\tilde{a})$, де $b_i = n^{-1} \sum_{j=0}^{n-1} \tilde{a}_j \omega^{-ij} \pmod{q}$ при $i = 0, 1, \dots, n-1$, 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 $c = a \cdot b \pmod{X^n + 1}$ 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 $a = (a_0, \dots, a_{n-1}), b = (b_0, \dots, b_{n-1}) \in Z_q^n$ - два вектори. Також визначимо $\hat{a} = (a_0, \psi a_1, \dots, \psi^{n-1} a_{n-1})$ і $\hat{b} = (b_0, \psi b_1, \dots, \psi^{n-1} b_{n-1})$. Негативно завернута a і b визначається як $c = (1, \psi^{-1}, \psi^{-2}, \dots, \psi^{-(n-1)}) \circ INTT(NTT(\hat{a}) \circ NTT(\hat{b}))$, 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 одиничних 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: $X' = X + Y \bmod p$
 $Y' = W(X - Y) \bmod p$

а) $X' \leftarrow X + Y$

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

в) $T \leftarrow X - Y$

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

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

е) $Y' \leftarrow U \bmod p$ (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 $W' = \lfloor W\beta/p \rfloor$

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) $T \leftarrow X - Y$
- г) *if* $T < 0$ *then* $T \leftarrow T + p$
- д) $Q \leftarrow \lfloor W'T/\beta \rfloor$
- е) $Y' \leftarrow (WT - Qp) \bmod \beta$
- 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 $TW \bmod p$ 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: $X' \equiv X + Y \bmod p, Y' \equiv W(X - Y) \bmod p$
 $X', Y' \in [0, 2p)$

- 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) $Y' \leftarrow (WT - Qp) \bmod \beta$

2.2.4 Зворотній алгоритм метелика

Розглянемо "зворотній алгоритм метелика" $\begin{bmatrix} X \\ Y \end{bmatrix} \rightarrow \begin{bmatrix} X + WY \\ X - WY \end{bmatrix}$. Це обернення звичайного алгоритму метелика (після заміни W на W^{-1} , і ділення на 2). Зворотній 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: $X' \equiv X + Y \bmod p, Y' \equiv W(X - Y) \bmod p$
 $X', Y' \in [0, 4p)$

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

б) $U \leftarrow WY \bmod p$ 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

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

Дано k кортежі $(a_1, t_1), \dots, (a_k, t_k)$, 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

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

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 $\epsilon \in R_q = \mathbb{Z}_q[x] / \langle x^n + 1 \rangle$, де розмірність n - потужність двох, тобто $n = 2^l$ для $l \in \mathbb{N}$.

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, \dots, a_k . Значення b_{GenA} вибираються експериментально, таким чином, щоб вони максимізували продуктивність на цільовій платформі Intel.

Пов'язані параметри та ймовірність прийняття. Значення L_S і L_E 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 налаштуємо їх параметри шляхом випробування різних значень для η_S та η_E в діапазоні $[2.0, \dots, 3.0]$ поки ми не знайдемо гарний компроміс між різними ймовірностями і, отже, тривалість виконання. Параметр B визначає інтервал випадкового многочлена u , і він визначається параметрами M і S як представлено у формулі 2.5.

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

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 $q \bmod 2n = 1$. 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 впливає, що теоретичний розмір секретного ключа задається як $n(k + 1)(\lceil \log_2(t - 1) \rceil + 1) + 2k + 320$ бітів.

3 ЦИФРОВИЙ ПІДПИС 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 ($e_0, e_1, \dots, e_{PARAM_K-1}$), a ($a_0, a_1, \dots, a_{PARAM_K-1}$))

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

ext [0],

s

```

ext [1],                                     e
ext [2],                                     a
ext [3],                                     у
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) m_{len} - 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
 $\text{seeds}[0], \text{seeds}[1], s, e := \text{qtesla_decode_sk}(\text{sk});$

2 $\text{input}[1] :=$ випадковий рядок довжиною 32 байти

3 $\text{input}[0] := \text{seeds}[1]$

4 знаходимо хеш для вхідного повідомлення
 $\text{input}[2, 3] := \text{qtesla_hash64}(m, \text{mlen})$

5 Відновлення поліному $a := \text{qtesla_gen_a}(\text{seeds}[0]);$

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

7 $v = y * a$

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

9 $c := \text{qtesla_gen_c}(c_); R/3$

10 $z := y + s * c$

11 $\text{success} := \text{qtesla_test_z}(z)$ Перевірка норми z

12 if $\text{success} = \text{ERROR}$ go to 6

13 $v := v - e * c$

14 $\text{success} = \text{qtesla_test_z}(v)$

15 if $\text{success} = \text{ERROR}$ go to 6

16 Перетворення цифрового підпису в рядок байтів
 $\text{sm}, \text{smlen} := \text{qtesla_encode_sig}(c, z)$

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`

4 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 -O3 -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., Rössler, 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 email 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., Rösler, 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.