Муниципальное бюджетное

общеобразовательное учреждение Гимназия

Новый Уренгой

ПРОЕКТНАЯ РАБОТА ПО ТЕХНОЛОГИИ

Профиль Информационная безопасность

Создание постквантового протокола   
обмена ключами шифрования

Исполнитель:

Учащийся 10 класса Брылёв Альберт

Руководитель работы:

Оборин Дмитрий Евгеньевич

*г. Новый Уренгой 2025*

**РЕФЕРАТ**

ПОСТКВАНТОВАЯ КРИПТОГРАФИЯ, KEM, РЕШЁТКИ, КВАНТОВЫЕ КОМПЬЮТЕРЫ, БЕЗОПАСНОСТЬ ДАННЫХ

Цель работы: Разработка постквантового криптографического протокола KEM на основе решёток.

В работе проведён обзор современных алгоритмов (Kyber, ML-KEM), изучены математические основы решёточной криптографии, рассмотрены угрозы квантовых вычислений.

Результат включает выбор инструментов для реализации прототипа протокола и его теоретическое обоснование.

Работа актуальна для обеспечения цифрового суверенитета и защиты данных в условиях развития квантовых технологий.

**Содержание**

**Термины, определения и сокращения4**

**Выбор, обоснование и актуальность6**

**Цели и задачи7**

**Анализ современных протоколов8**

**Анализ математических методов шифрования10**

**Методы шифрования на изогениях10**

**Методы шифрования на кодах, исправляющих ошибки11**

**Методы шифрования на многочленах12**

**Методы шифрования на решётках13**

**Методы шифрования на хэш-функциях14**

**Выбор и обоснование математической основы15**

**Анализ квантовых алгоритмов взлома16**

**Исследование криптографии на решётках17**

**Выбор инструментов для разработки проекта24**

**Создание протокола25**

**Математическая запись26**

**Анализ результатов экспериментов30**

**Реализация35**

**Заключение37**

**Ссылки и список литературы38**

**Приложение 140**

**1. Термины, определения и сокращения**

* KEM (Key Encapsulation Mechanism) — механизм инкапсуляции ключей, используемый для безопасной передачи криптографических ключей между участниками коммуникации.
* Кубит — квантовый бит, основная единица информации в квантовых вычислениях, которая может находиться в суперпозиции состояний.
* NIST (National Institute of Standards and Technology) — Национальный институт стандартов и технологий США, занимающийся стандартизацией в области криптографии.
* FIPS (Federal Information Processing Standards) — федеральные стандарты обработки информации, разрабатываемые NIST.
* ML-KEM (Module Lattice Key Encapsulation Mechanism) — механизм инкапсуляции ключей на основе модульных решёток, стандартизированный как FIPS 203.
* ML-DSA (Module Lattice Digital Signature Algorithm) — алгоритм цифровой подписи на основе модульных решёток, стандартизированный как FIPS 204.
* SLH-DSA (Stateless Hash-Based Digital Signature Algorithm) — алгоритм цифровой подписи на основе хэш-функций, стандартизированный как FIPS 205.
* LWE (Learning With Errors) — задача обучения с ошибками, используемая в криптографии на решётках.
* Module-LWE — модульная версия задачи LWE, используемая в протоколе Kyber.
* SVP (Shortest Vector Problem) — задача нахождения кратчайшего вектора в решётке.
* CVP (Closest Vector Problem) — задача нахождения ближайшего вектора в решётке.
* Центроцентрически симметричные распределение – распределение, у которого плотность вероятности симметрична относительно центральной точки (математического ожидания).
* Алгоритм Шора — квантовый алгоритм для факторизации чисел и решения задачи дискретного логарифма.
* Алгоритм Гровера — квантовый алгоритм для ускорения перебора в задачах поиска.
* QFT (Quantum Fourier Transform) — квантовое преобразование Фурье, используемое в алгоритме Шора.
* MPC-in-the-head — криптографический метод, используемый в протоколе Picnic.
* NTT (Number Theoretic Transform) — числовое теоретическое преобразование, используемое для оптимизации умножения многочленов.
* Rq = Zq[X]/(Xn + 1) — кольцо многочленов, используемое в протоколе Kyber.
* ECC (Elliptic Curve Cryptography) — криптография на эллиптических кривых.
* Pybind11 — библиотека для связывания C++ и Python.
* NumPy — библиотека для работы с массивами и матрицами в Python.
* SymPy — библиотека для символьных вычислений в Python.
* Git — система контроля версий.
* GitHub — платформа для хостинга кода и совместной работы.
* venv — инструмент для создания виртуальных окружений в Python.
* cProfile — модуль для профилирования производительности в Python.
* IDE (Integrated Development Environment) — интегрированная среда разработки.
* PyCharm — IDE для разработки на Python.

**2. Выбор, обоснование и актуальность**

На сегодняшний день криптографические протоколы обеспечивают безопасность передаваемой информации за счёт сложности современных алгоритмов шифрования. Однако, с появлением квантовых компьютеров ситуация может кардинально измениться. В настоящее время данные могут быть перехвачены, но расшифровать их без знания секретного ключа практически невозможно. При этом, как только квантовые компьютеры достигнут мощности в 3–4 тысячи кубит, они смогут эффективно взламывать существующие криптографические алгоритмы.

К примеру, по расчетам экспертов, RSA-2048 может быть успешно атакован квантовым компьютером, обладающим около 4000 кубит. Уже сегодня создан квантовый компьютер с 1000 кубитами. Предполагается, что к 2030 году появятся машины с необходимой мощностью для решения подобных задач.

Стоит также отметить недавние достижения компаний, таких как Google, которые разработали технологию компактных квантовых компьютеров, способных разместить 105 кубит на плате размером 4×4 см. Классические схемы обычно занимают десятки квадратных метров и высотой около 4х метром. Такой технологический скачок может ускорить развитие квантовых вычислений ещё больше, что в свою очередь подчеркивает необходимость своевременного перехода на постквантовые криптографические протоколы.

Почему нельзя откладывать разработку новых систем шифрования? Если ждать дальнейшего развития квантовых технологий, может наступить момент, когда будут недостаточно времени и ресурсов для создания и внедрения новых протоколов. Стандартизация первых постквантовых протоколов уже заняла около 6 лет (2016–2022), и в условиях стремительно меняющейся технологической среды задержки могут привести к тому, что обмен информацией останется уязвимым до тех пор, пока не будут разработаны и приняты новые стандарты безопасности.

Так же есть опасность «получи сейчас — расшифруй потом». То есть, злоумышленники могут перехватить информацию сейчас, а расшифровать потом. Это крайне опасно для государственной безопасности.

Переход на постквантовую криптографию является не только актуальной, но и крайне необходимой мерой для обеспечения безопасности данных в условиях стремительного развития квантовых вычислительных технологий, особенно в государственных и критически важных секторах.

**3. Цели и задачи**

Цель: разработка, анализ и реализация постквантового криптографического протокола типа KEM (Key Encapsulation Mechanism). В рамках работы предполагается:

1. **Исследование протоколов и математических основ:**Анализ существующих протоколов и их математических основ. Выбор оптимального метода шифрования и его глубокое изучение для последующей реализации.
2. **Исследование алгоритмов взлома и защиты от них:**

Анализ существующих алгоритмов взлома, их потенциал и возможные опасности. Как протоколы защищаются от них.

1. **Разработка эффективного механизма инкапсуляции ключей:**Разработка математических основ протокола KEM, который обеспечит надёжный обмен ключами, позволяя безопасно передавать криптографические ключи между участниками коммуникации.
2. **Практическая реализация и оценка протокола**Реализация прототипа предложенного решения с последующим проведением тестирования на устойчивость к атакам, а также сравнительный анализ с существующими постквантовыми решениями, такими как Kyber или Saber. Оценка будет проводиться по критериям, подобных NIST: безопасность, скорость и возможность внедрения.

Реализация протокола позволит не только повысить устойчивость обмена ключами, но и обеспечить практическую возможность перехода на постквантовые методы шифрования в государственных, коммерческих и других критически важных сферах.

**4. Анализ современных протоколов**

Для начала нужно разобраться с существующими протоколами.

На данный момент стандартизацией постквантовых протоколов занимается NIST, с 2016 года.

В 2022 году был проведён 3й раунд. Рассмотрим финалистов:

Для стандартизации были выбраны 4 протокола и 3 новых стандарта:

Crystals-Kyber (KEM на решётках) → FIPS 203 — ML-KEM

Crystals-Dilithium (подпись на решётках) → FIPS 204 — ML-DSA

Falcon (подпись на решётках)

SPHINCS+ (подпись на хэш-деревьях) → FIPS 205 — SLH-DSA

Все сравнения будут проходить с эталонами в своих категориях.

Конец третьего раунда:

* KEM:
  + Финалисты
    - Kyber (решётки) — эталон
    - NTRU (решётки) — медленней, не прошёл дальше
    - Saber (решётки) — проблема с доказательством, не прошёл дальше
    - Classic McEliece (коды) — безопасней, но медленней
  + Альтернативы
    - FrodoKEM (решётки) — медленней, не прошёл дальше
    - BIKE (коды) — быстрей, менее безопасный
    - HQC (коды) — немного уступает по безопасности и скорости
    - NTRUPrime (решётки) — медленней, не прошёл дальше
    - SIKE (изогении) — был взломан, не прошёл дальше
* Подписи
  + Финалисты
    - Dilithium (решётки) — эталон
    - Falcon (решётки) — эталон
    - Rainbow (многочлены) — был взломан, дальше не прошёл
  + Альтернативы
    - GeMSS (многочлены) — был взломан, дальше не прошёл
    - Picnic (MPC-in-the-head) — проблема с доказательством, не прошёл
    - SPHINCS+ (хэши) - эталон

**Российские постквантовые протоколы**

На данный момент у нас есть два сильных протокола:

* Шиповник — подпись на кодах
* Гиперикум — подпись на хэшах

Их разработка и стандартизация в рамках Технического комитета о стандартизации «Криптографическая защита информации» (ТК 26) Госстандарт.

Они активно внедряются в различные сферы и сотрудничают со множеством компаний, например МЦСТ (разработчик процессоров «Эльбрус»).

И есть 2 разрабатываемых KEM

* Кодиеум – коды, исправляющие ошибки
  + Пока открытый ключ занимает сотни килобайт, а время генерации ключа в сотни раз превышает существующие аналоги
* KEM на основе решёток
  + Разработка на ранней стадии, пока нет даже названия

5. Изучение математических методов шифрования

По порядку разберём основные направления:

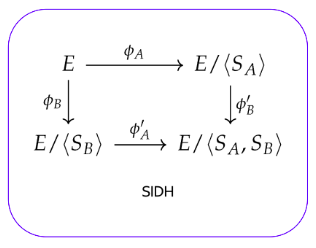
**Методы шифрования на изогениях**

Основная концепция:

* Используются эллиптические кривые и изогении между ними.
* Задача состоит в нахождении изогении между двумя заданными эллиптическими кривыми A и B.
* Основное преимущество метода: вычислительная сложность нахождения изогении при известных кривых.

Задача, лежащая в основе

* Поиск изогении (Isogeny problem):
* Даны две эллиптические кривые A и B.
* Необходимо найти изогению ϕ : A→B, которая является сложной задачей.



Основные работы:

* Схема Ростовцева и Столбунова
* Схема Куввеня
* Схема SIDH

SIDH уже взломана, но используется основой в данной категории криптографии.

Эллиптическая кривая – гладка алгебраическая кривая, заданная уравнением

Где a и b – коэффициенты, удовлетворяющие условию

Эллиптические кривые обладают групповой структурой, то есть на них можно определить операцию сложения точек, которая удовлетворяют условию группы.

Изогении – специальный вид отображения между двумя эллиптическими кривыми, которое является гомоморфизмом групп и сохраняет структуру кривых, то есть это морфизм эллиптических кривых, который переводит нейтральный элемент (бесконечно удалённую точку) одной кривой в нейтральный элемент другой кривой.

**Методы шифрования на кодах, исправляющих ошибки**

При использовании помехоустойчивого кодирования мы изменяем или добавляем данные к сообщению таким образом, чтобы оно стало устойчивым к ошибкам в канале связи. Это позволяет получателю восстановить исходное сообщение, несмотря на помехи.

Проблема декодирования случайного кода:

* Структура кодов:

Коды строятся на основе определенной структуры, что позволяет корректно их декодировать.

* Случайная генерация:

Если код сгенерирован случайно, его декодирование становится невозможным, так как отсутствуют правила для его расшифровки.

Применение в криптографии:

Сообщение маскируется под случайно сгенерированный код, к которому добавляется специально созданная ошибка. Это делает расшифровку невозможной для злоумышленника, так как:

* Отсутствует структура: злоумышленник не может определить схему декодирования.
* Добавленная ошибка: усложняет процесс расшифровки.

Криптосистема Мак-Элиса использует помехоустойчивые коды для шифрования. Сообщение кодируется через порождающую матрицу G′ = SGP, где:

* G — порождающая матрица.
* S и P — матрицы для добавления "шума".

Зашифрованное сообщение (шифротекст) c формируется как: c = mG′ + e

где e — вектор ошибки.

Основные работы:

* Криптосхема Мак-Элиса
* Криптосхема Нидеррайтера
* ID-схема Штерна

**Методы шифрования на многочленах**

1. Основная идея:

* В основе криптографических схем лежат многочлены от многих переменных с коэффициентами из конечных полей.
* Используются нелинейные системы уравнений, решение которых является вычислительно сложной задачей.

2. Сложные задачи в основе:

* Задача MQ (Multivariate Quadratic problem):
* Найти решение для системы квадратичных многочленов над конечным полем.
* Задача доказана как NP-трудная, что делает её идеальной для криптографии.

3. Применение:

* Такие криптосистемы часто используются для:
* Электронных подписей.
* Аутентификации.
* Генерации ключей.

4. Алгебраическая основа:

* Применяется сложная алгебра над конечными полями.
* Обеспечение безопасности зависит от невозможности эффективно решать системы квадратичных уравнений с большим количеством переменных.



Основные работы:

* Подпись Матсумото-Имаи

**Методы шифрования на решётках**

1. Геометрия решёток и аналогия с кодами:

* Решётка в геометрическом смысле — это множество точек, равномерно распределённых в пространстве.
* Она задаётся базисом, и от его качества зависит сложность работы с решёткой.
* Если базис плохой (например, с большими коэффициентами), найти близкую к заданной точку решётки или работать с ней сложно.

2. Задача декодирования:

* Сообщение кодируется вектором v = Bm, где B— базис решётки, а m — сообщение.
* Добавляется ошибка e, и результатом шифрования становится

c = v + e.

* Для дешифровки используется "хороший" (секретный) базис R, который позволяет корректно восстановить сообщение.
* Если у злоумышленника есть только "плохой" базис, то без знания секретного базиса декодирование практически невозможно.

3. Основные задачи на решётках:

* Нахождение кратчайшего вектора (Shortest Vector Problem, SVP): Задача нахождения минимального ненулевого вектора в решётке.
* Нахождение ближайшего вектора (Closest Vector Problem, CVP):Определить ближайшую точку решётки к заданной.
* Эти задачи сложны с вычислительной точки зрения и служат основой криптографии.

4. Особенности решёточных криптосистем:

* "Зашумлённый" базис делает невозможным решение CVP без дополнительной информации.

Основные работы:

* Односторонняя функций Айтая
* Схема NTRU
* Схема шифрования LWE Регева
* Подпись Любашеввского

**Методы шифрования на хэш-функциях**

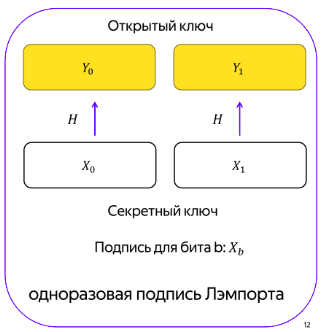
Одноразовая подпись Лэмпорта

* Секретный ключ - случайные значения
* Открытый ключ - хэш секретного ключа

Когда нам нужно подписать какой-то бит мы публикуем прообраз соответствующего хэша

Остальные подходы:

Случайно генерируем случайные значения и хешируем n раз. Что бы подписать сообщение b мы хэшируем log2 b до n.

Что бы проверить мы дохэшируем и проверяем, что действительно так и было.

Так же существуют «**экзотические**» решения, например MPC-it-the-Head, но здесь мы не будем его рассматривать, так как будут большие проблемы с доказательством.

**6. Выбор и обоснование математической основы**

Проанализировав существующие протоколы, их плюсы и недостатки я остановился на решётках.

Моё предпочтение в пользу криптографии на решётках обусловлено не только теоретической надёжностью, но и прагматичными соображениями, связанными с технологическим суверенитетом и долгосрочной безопасностью

1. Теоретическая фундаментальность и проверенная стойкость

Криптография на решётках — одна из наиболее популярных и перспективных направлений. Сейчас лидирующие протоколы в основном на решётках.

* + Увеличить точность доказательства криптостойкости:
  + Опираться на готовые стандарты

1. Импортозамещение и технологическая независимость

В России сегодня доминируют постквантовые протоколы-подписи, а KEM только остаётся в разработках

* + Уникальность разработки

Создание отечественного KEM на решётках позволит закрыть недостающее звено в линейке российских постквантовых криптографических протоколов.

* + Минимизация внешних рисков

Зарубежные решения, даже с открытым кодом, могут нести риски преднамеренного вмешательства и усложнения или даже запрет на внедрения в наши системы.

1. Будущая адаптивность и экосистемный вклад

Решётки обладают гибкостью для модификаций и всё время дорабатываются, что критично в условиях быстро меняющихся угроз. Кроме того, разработка российского стандарта KEM:

* + Стимулирует развитие отрасли
  + Укрепит экспортный потенциал

Выбор решёток — это не просто следование глобальным трендам, а осознанная стратегия, сочетающая научную обоснованность с национальными интересами. Разрабатывая KEM мы не только защищаем данные от квантовых угроз, но и создадим инфраструктуру, свободную от внешних рисков, что соответствует курсу на цифровой суверенитет.

**7. Изучение квантовых алгоритмов взлома**

Наиболее известными являются алгоритмы Шораи Гровера, которые способны подорвать безопасность асимметричных и симметричных протоколов соответственно. Рассмотрим их подробно.

1. **Алгоритм Шора**: угроза асимметричной криптографии

**Цель:** факторизация больших чисел и решение задачи дискретного логарифма, что ставит под угрозу RSA, ECC и другие системы с открытым ключом.

Принцип работы**:**

1. Классическая часть

* Задача факторизации сводится к поиску периода функции f(x) = ax mod N, где N — число для разложения, a — случайное число, взаимно простое с N.
* Позволяет факторизовать число N за полиномиальное время (O(log3 N)), используя O(log N) кубит.
* Если период r найден и чётен, то делители N вычисляются как gcd(ar/2 ± 1, N)

**2. Квантовая часть**

* Используются два квантовых регистра. Первый создаёт суперпозицию всех возможных значений *x*, второй вычисляет *f*(*x*).
* Применяется квантовое преобразование Фурье (QFT) к первому регистру, что позволяет выделить период *r* из суперпозиции состояний.
* Измерение регистра даёт значение, связанное с периодом *r*, который затем используется для факторизации.

Пример угрозы:

* Для взлома RSA-2048 потребуется около 4000 кубит и не на много больше времени, чем создание ключа, то есть около секунды.

Улучшения:

* В 2023 году Одед Регев предложил модификацию алгоритма, сокращающую количество квантовых вентилей за счёт использования многомерной геометрии.
* Исследователи MIT оптимизировали подход Регева, применяя числа Фибоначчи для расчёта экспонент, что снизило требования к памяти.

**2. Алгоритм Гровера: угроза симметричной криптографии**

**Цель:** ускорение перебора в задача поиска, что снижает безопасность симметричных алгоритмов (AES, DES).

Принцип работы**:**

* Для поиска элемента в неупорядоченной базе данных из *N* элементов классический алгоритм требует *O*() операций, а алгоритм Гровера — *O*()
* Использует квантовую амплитудную интерференцию: многократное применение оператора оракула и диффузии для усиления амплитуды искомого состояния.

Пример угрозы:

* Для AES-256 время взлома сокращается с 2256 до 2128 операций

Улучшения:

* Был предложен алгоритм BHT, что ускоряет перебор до *O*().

3. Другие алгоритмы и направления

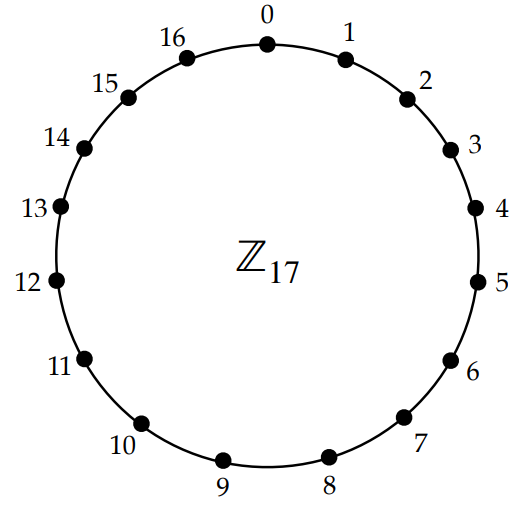
* Алгоритм Регева (2023): Улучшение Шора за счёт обобщения на многомерные решётки, что снижает сложность вычисления.
* Атака на эллиптические кривые: Алгоритм Шора адаптирован для решения задачи дискретного логарифма на эллиптических кривых (ЕСС) что угрожает ГОСТ 34.10-2018 и аналогичным стандартам.

То есть асимметричные протоколы может эффективно взламывать только алгоритм Шора и его модификации. Значит доказательство криптостойкости должно основываться на алгоритме Шора.

**8. Углубленное изучение криптографии на решётках**

Для начала изучим как именно работают алгоритмы у других протоколов.

Для примера возьмём Kyber:



**1. Основные компоненты и параметры**

* **Параметры:**

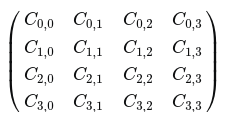
Кольцо многочленов:

    Работает с кольцом Rq=Zq[X] / (Xn+1). Обычно выбирают

    n = 256 — степень многочленов.

q = 3329 — модуль (простое число).

* **Матрица A:**

Фиксированная (или генерируемая по seed) матрица многочленов размером k×k где k зависит от выбранного уровня безопасности, например, для Kyber-512 используется k=2, для Kyber-768 — k=3, для Kyber-1024 — k=4). Каждый элемент матрицы — многочлен из Rq.

* **Секреты и шум:**

    Секретные вектора и ошибки задаются как векторы многочленов, коэффициенты которых выбираются согласно небольшому дискретному распределению (например, центроцентрически симметричное распределение с малой дисперсией, близкое к биномиальному).

**Идея протокола:**

Основная сложность, на которой базируется безопасность, — задача Module Learning With Errors (Module-LWE). Она сводится к поиску секрета s по уравнению

b = A ⋅ s + e (в Rq),

где e — шум, выбранный случайно из распределения с малыми значениями. Из-за наличия шума восстановить s из A и b практически невозможно.

**2. Фазы алгоритма**

**2.1 Генерация ключей (KeyGen)**

**Цель: Сгенерировать пару ключей:**

* Открытый ключ pk — используется для шифрования (инкапсуляции).
* Секретный ключ sk — используется для дешифровки (деинкапсуляции).

**Пошаговое описание:**

1. Генерация секретного вектора s и шумового вектора e:

Для каждого из k элементов генерируются многочлены с малыми коэффициентами, выбранными из фиксированного распределения например, биномиального распределения).

2. Формирование матрицы A:

Матрица A может быть определена фиксированным seed, от которого затем псевдослучайно генерируются её коэффициенты, либо может передаваться как часть системных параметров.

3. Вычисление вектора b:

Для открытого ключа вычисляется

    b = A ⋅ s + e

    Операция умножения и сложения выполняется в кольце Rq (то есть все коэффициенты приводятся по модулю q).

4. Формирование ключей:

* Открытый ключ: pk = (A, b).
* Секретный ключ: sk = s (часто в реальной реализации дополнительно сохраняют информацию для защиты от атак с адаптивным выбором сообщений или побочных каналов).

**2.2 Инкапсуляция (Encaps)**

**Цель: Отправитель генерирует зашифрованную капсулу, которая содержит зашифрованный симметричный ключ, и отправляет её получателю.**

**Пошаговое описание:**

1. Генерация случайного ключа и выбор шумовых многочленов:

* Выбирается случайное сообщение m (например, бинарный вектор фиксированной длины, который потом может использоваться как симметричный ключ или служить для генерации ключа).
* Генерируются новые шумовые векторы r, e1 и e2​ (каждый из них — вектор из k многочленов с малыми коэффициентами).

2. Вычисление промежуточных значений:

* Вычисляется u = AT ⋅ r + e1​. Здесь AT - транспонированная матрица A.
* Вычисляется v = bT ⋅ r + e2​. Затем к v добавляется сообщение m с применением процедур квантования или кодирования, чтобы обеспечить корректное восстановление m при деинкапсуляции.

3. Формирование капсулы:

    Капсула c состоит из двух частей: c = (u, v)

**2.3 Декапсуляция (Decaps)**

**Цель: Получатель, обладая секретным ключом s, извлекает исходное сообщение m из капсулы c = (u, v).**

**Пошаговое описание:**

1. Вычисление промежуточного значения:

Используя свой секрет s, получатель вычисляет

    v′ = uT ⋅ s

    где uT — транспонированное представление вектора u (если u представлен в виде вектора многочленов, операция сводится к поэлементному умножению с последующим суммированием).

2. Извлечение сообщения:

После вычисления v′ происходит операция обратного квантования (decoding) с

  m′ = decode(v − v′)

Если ошибок не слишком много (что гарантируется корректно выбранными параметрами распределения шума), то m′ совпадёт с исходным m.

**3. Пример работы на упрощённом варианте**

Для иллюстрации рассмотрим упрощённый пример (с меньшими размерами параметров):

Предположения:

* Пусть n = 4 и q = 17 (для простоты примера; в реальности n = 256 и q=3329).
* Пусть k = 2.

Генерация ключей

1. Выбираем секрет s и ошибку e:

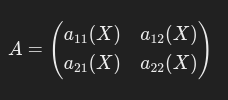
    Например, пусть:

* + s1(X) = 1 + 0X + 2X2 + 0X3
  + s2(X) = 0 + 1X + 0X2 + 1X3

        (коэффициенты выбираются из маленького множества, скажем, {−2,−1,0,1,2}).

2. Матрица A:

    Пусть для простоты:

 где, например,

* a11(X) = 3 + 5X + 0X2 + 2X3
* a12(X) = 1 + 2X + 4X2 + 0X3
* a21(X) = 2 + 1X + 3X2 + 3X3
* a22(X) = 0 + 4X + 1X2 + 2X3

3. Вычисляем b = A ⋅ s + e:

Пусть шум e также состоит из двух многочленов с малыми коэффициентами, например:

* + e1(X) = 1 + 0X + (-1)X2+ 0X3
  + e2(X) = 0 + 1X + 0X2+ (-1)X3

    Для первого элемента b1(X):

* Вычисляем a11 \* s1(X) и a12 \* s2(X) по модулю 17 с учётом свёртки по X4 + 1.

Аналогично для b2(X).

(Подробные вычисления выполняются по правилам умножения многочленов в кольце Rq.)

4. Получаем открытый ключ:

    pk = (A, b).

**Инкапсуляция**

Предположим, отправитель выбирает сообщение m (например, 4‑битное представление, закодированное в многочлене).

Пусть:

* m после кодирования даёт многочлен m(X) = 1 + 0X + 1X2+ 0X3

Отправитель генерирует шумовые многочлены r, e1 (вектор длины 2) и e2 (один многочлен). Затем:

1. Вычисляет u = AT \* r + e1

Если r = (r1(X), r2(X)), то

u1(X) = a11(X) \* r1(X) + a21(X) \* r2(X) + e1,1(X)

и

u2(X) = a12(X) \* r1(X) + a22(X) \* r2(X) + e1,2(X)

2. Вычисляет v = bT \* r + e2 + encode(m).

Например,

v(X) = b1(X) \* r1(X) + b2(X) \* r2 + e2(X) + encode(m(X))

Капсула c = (u, v) отправляется получателю.

**Декапсуляция**

Получатель, зная секрет s = (s1, s2) выполняет следующие шаги:

1. Вычисляет v′ = u1(X) \* s1(X) + u2(X) \* s2(X)

Вычисляет разность v(X) — v′(X), которая должна приблизительно равняться encode(m(X)) (так как шумы, внесённые на этапах, малы).

Применяет функцию декодирования decode(⋅), чтобы восстановить исходное сообщение m.

Если параметры и распределения шума выбраны корректно, вероятность ошибки декодирования очень мала.

**4. Заключительные замечания**

**Кодирование и декодирование:**

Для того чтобы сообщение m было корректно восстановлено, Kyber применяет процедуры квантования. Часть коэффициентов многочлена делят на фиксированный делитель и сравнивают с пороговыми значениями для определения битов m. Эти процедуры могут быть довольно детализированы в спецификации.

**Оптимизации:**

Реальные реализации Kyber используют оптимизированные алгоритмы для умножения многочленов (например, с использованием Number Theoretic Transform, NTT) для повышения производительности.

**Безопасность:**

Защита основана на сложности задачи MLWE. Даже при наличии квантовых компьютеров, эффективное решение этой задачи остаётся маловероятным при корректном выборе параметров.

**9. Выбор инструментов для разработки проекта**

1. Язык программирование
   * Python 3.12
     + Причины выбора:
       - Быстрая разработка
       - Богатая экосистема
       - Сообщество и документация
   * C++ (планируется)
     + Цели перехода:
       - Увеличение производительности
       - Управление памятью
       - Интеграция с аппаратным обеспечением
2. Библиотеки
   * NumPy
   * SymPy
   * Zeph1rr-kuznechik
   * Matpotlib
   * Random
   * Socket
   * NetCat
3. Инструменты разработки:
   * IDE - PyCharm
   * Система контроля версий — Git + GitHub
4. Инфраструктура и окружение
   * Виртуальное окружение venv
   * Мониторинг производительности — cProfile
5. Критерии отбора
   * Скорость разработки и производительность: Python для создания прототипа, C++ для финальной версии.
   * Совместимость: использование Pybind11 обеспечит плавный переход между языками

**10. Создание протокола**

LWR и LWE — два тесно связанных криптографических предположения, но LWR предлагает ряд улучшений в эффективности и практичности. Вот основные преимущества протоколов на LWR:

1. Проще в работе
   * В LWE генерируется шум, а в LWR используется округление. Это ускоряет вычисления
2. Меньше ‘лишних’ данных
   * LWR сжимает информацию за счёт округления. Это экономит память и трафик, что важно для микропроцессоров
3. Стабильность
   * В LWE шум может накапливать и ломать алгоритм
4. Безопасней против некоторых атак
   * Некоторые атаки, например анализ шума, не работают против LWR из за детерминированного округления

Разработка криптографического протокола включала в себя несколько ключевых этапов:

* Выбор параметров
* Построение механизмов округления и восстановления ключа
* Тестирование корректности работы схемы

Основой послужили принципы постквантовой криптографии, в частности, механизмы, используемые в Saber.

Главная идея – оптимизация вычислений для применения NNT.

**Формирование ключей**

Генерация ключей в протоколе осуществляется следующим образом:

Открытый ключ – многочлен A, выбранный случайно из

Секретный ключ – многочлен s, выбираемый из ограниченного множества например, {0, 1}n

Секретный ключ используется для вычисления зашифрованного сообщения и его последующей расшифровки.

**Механизм округления и восстановления ключа**

В основе схемы лежит округление значений, которое позволяет корректно восстанавливать общий ключ у обеих сторон

Для этого:

* В процессе шифрования данные масштабируются и округляются по формуле [x / d] mod p, где d = q / p
* На стороне расшифровки применяется механизм коррекции ошибок, позволяющий компенсировать отклонения, вызванными округлением

**Операции с многочленами**

Протокол использует операции умножения многочленов по модулю xn + 1. В текущей реализации применяется наивный метод умножения, который,   
несмотря на корректность, требует оптимизации.

Например можно использовать быстрое преобразование Фурье (NTT), что позволит работать с большими числами без значительной задержки

**Математическая запись**

**Параметры системы**

q = 27 – модуль для арифметики в кольце

p = 23 – новая точность после округления

n = 24 – степень полиномов

d = q / p + 1 = 24 + 1 – фактор деления для округления

offset – смещение, равен d / 2

threshold – пороговое значение, используемое при вычислении подсказки

Сейчас выбраны маленькие коэффициенты для прототипа

Мы работает в кольце

То есть с многочленами вида a(x) =

При этом выполняется условие xn = -1

**Умножение многочленов**

Для двух многочленов

И

Их произведение вычисляется по формуле

c(x) = a(x) \* b(x) mod (xn + 1)

То есть коэффициенты ck задаются так

На практике для каждого коэффициента сk справедливо:

Если i + j < n: ci+j +=

Если i + j n: ci+j -=

Так как xn ≡ -1 mod (xn + 1) все операции (сложение, вычитание) берутся по модулю.

**Округление коэффициентов**

Для каждого коэффициента x , выводится функция округления с   
параметром смещения offset

Round(x) =

Так как offset = d / 2 (или ближайшее целое), это обеспечивает симметричное округление.

Применяя округление к каждому коэффициенту многочлена   
a(x) = (a0, a1, ..., an-1)

Получаем:

**Вычисление hint (подсказки)**

Для каждого коэффициента x из некоторого многочлена вычисляется  
значение подсказки:

То есть берётся остаток от деления x на d, и если он не меньше порога   
threshold, устанавливается единица.

**Корректировка на стороне декодирования**

При декодировании используется функция «reconcile», которая сдвигает коэффициент на ±1 в зависимости от подсказки. Обозначим wi = Round(ui⋅si)*wi* и hint = *hi*​. Тогда:

Это даёт более точное значение

**Кодирование и декодирование сообщения**

Задаём сообщение m = (m0, m1, ..., mn-1), где mi {0, 1}.

**Кодирование**

То есть, если = 1, коэффициент равен p / 2, а если = 0, то 0.

**Декодирование**

Где – коэффициент из расшифрованного полинома.

**Генерация ключей**

1. Выбор случайного многочлена a(x):  
   Генерируется a(x) с коэффициентами, равномерно распределёнными по
2. Секретный многочлен s(x)  
   Выбирается s(x) с коэффициентами из {0, 1}
3. Вычисление произведения c(x) = a(x) \* s(x) mod (xn + 1)
4. Округление: b(x) = Round(c(x)), то есть округляем покоэффициентно и берём результат в Zp.

Pk = (a(x), b(x)), sk = s(x)

**Инкапсуляция**

Пусть pk=(a(x),b(x))pk=(*a*(*x*),*b*(*x*)).

1. Случайное сообщение m из {0,1}n.
2. Кодируем: menc(x) = encode(m).
3. Эфемерный секрет s′(x), где коэффициенты ∈{0,1}.
4. Вычисляем  
   u′(x) = a(x) s′(x) mod (xn+1),  
   u(x) =  Round(u′(x)).
5. Подсчитываем b′(x) = b(x) ⋅ s′(x). Затем  
   h(x) = hint(b′(x)) и vround(x) = Round(b′(x)).
6. Формируем  
   v(x) = vround(x) +  menc(x)   mod   p.
7. Шифротекст  
   ct = (u(x), v(x), h(x)).
8. Общий ключ:   
   K = Hash(Serialize(m))

(Serialize(m) — это преобразование в двоичную/байтовую строку.)

**Декапсуляция**

Пусть получатель имеет sk = s(x) и получает ct = (u(x), v(x), h(x)).

1. **Промежуточное произведение**w′(x)=u(x) s(x) mod (xn+1), w(x)=Round(w′(x)).
2. **Корректировка** с помощью hint:  
     =  reconcile(wi,  hi),  
   где hi​ – соответствующий коэффициент hint(x).
3. **Извлечение сообщения**:  
    =  vi  −     mod   p.
4. **Декодирование**:
5. **Общий ключ:**K′ = Hash(Serialize())

Если ошибок округления нет(или они компенсированы подсказками),то K′ = K

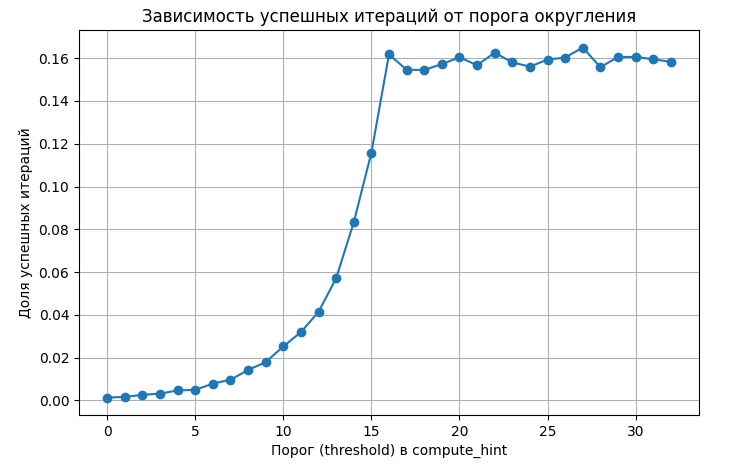
**Анализ результатов экспериментов**

### **1. Зависимость успешных итераций от порога округления (Threshold)**

На первом графике (см. рис. 1) приведена зависимость доли успешных итераций (Success Rate) от значения порога threshold, используемого в функции compute\_hint. В ходе эксперимента threshold изменялся от 0 до 32, а остальные параметры (q, p, n) были зафиксированы.

* **Рост при малых threshold**При threshold = 0 наблюдается низкая доля успешных итераций (0.14%), что объясняется тем, что подсказка (hint) практически не даёт информации о корректировке. С увеличением threshold\ вплоть до 16 доля успешных итераций возрастает (с 0.14% до 16.18%).  
  Это говорит о том, что при слишком низком пороге мы «слишком часто» не корректируем значение, а при слишком высоком – корректируем избыточно. Между этими крайностями формируется оптимальный диапазон, где подсказка даёт наилучший эффект.
* **Поведение при threshold > 16**Начиная примерно с threshold=16, кривая выходит на «плато» в районе 15–16% успеха, но при этом продолжает колебаться. Самая высокая точка (глобальный максимум) зафиксирована при threshold = 27, где Success Rate достигает 16.50%. Это указывает на то, что даже при значениях порога, больших d/2, можно получить небольшой выигрыш, но в целом после 16 эффект уже не столь выраженный, и кривая колеблется около одной и той же области значений.
* **Суммарные наблюдения**
  + **Наибольшее значение успеха (экстремум)**:   
    16.50% threshold = 27.
  + **Локальный максимум** в точке threshold = 16 равен 16.18%.
  + При слишком малых значениях порога (threshold < 5)   
    успех не превышает 1%.
  + Наиболее «выгодная» зона threshold лежит примерно в диапазоне 15–20, где Success Rate стабильно находится в районе 15–16%.

Таким образом, из графика видно, что выбор порога напрямую влияет на качество восстановления ключа. Слишком низкие значения дают недостаточную коррекцию, а слишком высокие – перенасыщенную. Оптимальное значение threshold в рамках данных параметров находится в районе d/2 или немного выше.

Рис. 1

### **2. Влияние параметров q, p и n**

На втором наборе графиков (см. рис. 2) исследуется, как изменение основных параметров протокола отражается на доле успешных итераций:

1. **Влияние q**

* На графике видно, что при увеличении q (например, с 128 до 1000 и выше) доля успешных итераций снижается.
* Низкое q (например, 128) даёт более высокий успех (по сравнению с большими q), так как «зона» для шума меньше, и округление происходит более предсказуемо. Однако это упрощает схему и может ухудшать криптографическую стойкость.

1. **Влияние p**

* Аналогично, при увеличении p (например, от 8 до 64) наблюдается снижение доли успешных итераций.
* При маленьком p (например, 8) округление даёт более грубую квантизацию, и, парадоксально, в тестовых условиях это может повысить вероятность совпадения у обеих сторон. Однако в реальных условиях такое уменьшение p может затруднить надёжное восстановление сообщения при большом шуме.

1. **Влияние n**

* Чем больше n, тем сложнее обеспечить высокую точность восстановления (увеличивается число коэффициентов, суммарная ошибка растёт). На графике видно, что при увеличении n доля успешных итераций падает.
* При маленьком n (например, 16) меньше коэффициентов, и восстановление ключа проще. Но при этом криптографическая стойкость такой схемы тоже снижается.

**Лучшие результаты** в серии экспериментов показала комбинация q=128, p=8, n=16, при которой доля успешных итераций достигла 35.10%.  
Это говорит о том, что в данных условиях (с минимальным размером полинома и относительно небольшими модулями) система может работать с высокой вероятностью корректного восстановления, хотя, разумеется, такие параметры не обеспечивают реальную криптостойкость.



Рис. 2

## **Выводы**

1. **Выбор порога**График зависимости от threshold демонстрирует, что существует «золотая середина»: слишком маленький или слишком большой порог даёт низкую точность восстановления, а оптимальный диапазон лежит около d/2. Максимальное значение (16.50%) получено при   
   threshold = 27, однако уже при threshold=16 успех достигает 16.18%, что ненамного меньше.
2. **Зависимость от q, p, n**

* С увеличением q и p доля успеха уменьшается, так как возрастает пространство для шума и сложнее добиться точного совпадения.
* Увеличение n также ухудшает результат, поскольку увеличивается число коэффициентов и суммарная ошибка.
* Максимальную долю успешных итераций (35.10%) дали минимальные параметры q = 128, p = 8,n = 16.

1. **Практическая значимость**Полученные результаты свидетельствуют, что при выборе параметров нужно искать компромисс между криптографической стойкостью (требует больших q и n) и корректностью восстановления (которая выше при меньших значениях). В реальных схемах, таких как Saber, параметры значительно больше, а механизм округления и распределение секретных коэффициентов тонко настраиваются для обеспечения и   
   высокой безопасности, и надёжного восстановления ключа.

Таким образом, проведённые эксперименты дают наглядное представление о том, как основные параметры и порог подсказки влияют на точность (Success Rate) протокола. Для практического применения необходимо дальнейшее масштабирование и более сложные методы оптимизации, однако полученные данные позволяют выявить основные тенденции и подобрать разумные тестовые настройки.

**Какие значения параметров q, n нужны?**

1. **Параметр q** — это модуль для арифметики в кольце, в котором выполняются операции над полиномами. Он должен быть достаточно большим, чтобы обеспечить криптографическую стойкость, но не слишком большим, чтобы не создавать чрезмерные вычислительные затраты. Обычно для устойчивости к квантовым атакам q выбирают значения от 212 до 216, а для сильной защиты — от 216 до 232.
2. **Параметр n** — это степень полинома, то есть количество коэффициентов в многочлене. Для обеспечения хорошей безопасности следует выбирать n не менее 512 или 1024. Маленькие значения n, такие как 8 или 16, делают систему уязвимой для атак методом полного перебора или использования ошибок округления.
3. **Параметр p** — это точность после округления. Обычно выбирается относительно небольшим значением, например, 32 или 64, в зависимости от требований к точности и вычислительной сложности. Большие значения p увеличивают сложность, но при этом затрудняют правильную декодировку сообщения.

**Как зависит криптостойкость от этих параметров?**

1. **Зависимость от q**: Чем больше значение q, тем больше возможных значений для коэффициентов полиномов, что усложняет атакующим задачу восстановления секретного ключа с помощью подбора. Это также увеличивает сложность атак с ошибками, таких как атаки на основе ошибок округления или атак методом полного перебора.
2. **Зависимость от n**: Увеличение n увеличивает размер пространства возможных полиномов, что значительно усложняет задачу для атакующего, поскольку увеличение n требует экспоненциального роста вычислительных мощностей для атаки.
3. **Зависимость от p**: Значение p определяет точность вычислений и влияние ошибок округления. Малые значения p делают алгоритм более уязвимым к атакам, связанным с ошибками округления, а также ускоряют вычисления. Однако, чем больше p, тем выше криптографическая стойкость, но и возрастает вычислительная сложность.

**Оценка времени работы:**

* Время работы алгоритма зависит от операций с полиномами (умножение, округление, вычисление подсказок), что ведет к повышенной вычислительной сложности с ростом n и q.
* С увеличением размера n или *q*, время работы растет экспоненциально, поскольку количество операций умножения полиномов и их округления увеличивается. Примерно, сложность для вычислений с полиномами может быть оценена как O(n2), так как операция умножения полиномов требует O(n2) операций.

На выполнение 10000 итераций алгоритма в среднем занимает 0,59 секунд.

**11. Реализация**

Для прототипа был выбран язык программирования Python. В данном случае выбран для удобства разработки и наглядности.   
Для высокопроизводительных решений на практике могут использоваться более быстрые языки (C/C++/Rust) и оптимизированные библиотеки, в том числе с реализацией быстрых преобразований Фурье (NTT).

### **Основная структура кода**

1. **Объявление параметров**В начале программы задаются основные параметры: q, p, n а также вычисляется вспомогательный коэффициент d=q / p​. Эти параметры определяют размерность кольца многочленов и масштаб округления.
2. **Функции для работы с многочленами**
   * **Умножение** в кольце .
   * **Округление** коэффициентов и вычисление **hint**.
   * **Кодирование/декодирование** сообщения.
3. **Генерация ключей (KeyGen)**Функция генерирует случайный многочлен a(x) и секретный многочлен s(x), после чего вычисляет b(x) путём умножения a(x) ⋅ s(x) и последующего округления.
4. **Инкапсуляция (Encapsulation)**
   * Выбирается случайное сообщение и кодируется в полином menc(x).
   * Генерируется эфемерный секрет s′(x), с помощью которого вычисляются u(x) и v(x), а также подсказка hint.
   * Результат ct=(u,v,hint) возвращается как шифротекст, а общий ключ K получается из хэша сериализованного сообщения.
5. **Декапсуляция (Decapsulation)**

* На стороне получателя, используя секретный ключ s(x), вычисляется w(x) из u(x) ⋅ s(x).
* Применяется функция **reconcile** с учётом подсказки hint, что помогает корректно восстановить menc(x).
* После декодирования получается исходное сообщение, из которого берётся общий ключ K′.
* Если схема работает корректно, K′ совпадает с K

### **Тестирование и экспериментальная проверка**

Для оценки корректности и изучения характеристик протокола проводились эксперименты с различными параметрами:

**1. Запуск на локальном компьютере**

* Проверка базовой функциональности (генерация ключей, шифрование и расшифрование).
* Небольшие тестовые прогоны, в ходе которых сравнивали совпадение общего ключа на стороне отправителя и получателя.

**2. Использование суперкомпьютера РАН**

* Для более масштабных экспериментов, включая варьирование параметров q, p, n и порога threshold, и проверки связи программа была запущена на суперкомпьютере Российской академии наук.

**3. Сбор и визуализация данных**

* По итогам каждого эксперимента собирались метрики: доля успешных итераций, время выполнения, ошибки декодирования.
* Данные обрабатывались с помощью библиотеки matplotlib для построения графиков и последующего анализа.
* Результаты были сопоставлены с теоретическими ожиданиями, а также с аналогичными схемами (например, Saber).

### **Особенности и рекомендации**

* **Оптимизация**: текущая реализация на Python подходит для прототипирования, однако для реального использования потребуются оптимизированные алгоритмы умножения многочленов (NTT) и более эффективные операции округления.
* **Безопасность**: выбранные тестовые параметры (q, p, ) не обеспечивают постквантовую стойкость; они нужны лишь для иллюстрации. В практических схемах (например, Saber) применяются существенно большие значения и более сложные распределения шума.
* **Масштабируемость**: благодаря тому, что код разбит на функции, расширение протокола под более крупные параметры или другие механизмы распределения секретов (например, гауссовское) не требует кардинальной переработки структуры.

.

**12. Заключение**

В ходе работы был разработан и протестирован прототип криптографического протокола, основанного на идеях схемы LWR с заимствованиями из Saber. В рамках исследования были выполнены следующие этапы:

* **Математическое обоснование протокола:**Представлены основные алгоритмические и теоретические моменты, связанные с операциями в кольце многочленов, механизмами округления и восстановлением общего ключа. Детально описаны формулы, лежащие в основе схемы, а также методы вычисления подсказки (hint) для коррекции ошибок.
* **Реализация прототипа:**Протокол был реализован на языке Python с использованием стандартных библиотек для генерации случайных чисел, работы с матрицами и визуализации данных. Проведённое тестирование, включая запуск на суперкомпьютере Российской академии наук, позволило собрать статистику по успешности восстановления ключа и изучить влияние различных параметров.
* **Экспериментальное исследование:**Были проведены эксперименты, позволяющие установить зависимость успешности восстановления общего ключа от значений порога округления и основных параметров q, p и n. Результаты показали, что даже при тестовых, упрощённых параметрах прототип демонстрирует корректное функционирование, хотя для практической криптостойкости необходимо масштабирование и оптимизация.

Разработанный прототип является доказательством концепции, подтверждающим возможность создания постквантового протокола на базе LWR-подхода. Полученные результаты демонстрируют основные тенденции: правильный выбор параметров и механизма округления существенно влияет на корректность восстановления ключа. Однако, текущая реализация использует тестовые параметры, что обеспечивает лишь работоспособность, а не реальную криптографическую стойкость.

В дальнейшем планируется:

* Увеличение значений q и n для обеспечения надежной постквантовой защиты;
* Оптимизация алгоритмов умножения многочленов (например, с использованием NTT);
* Доработка механизмов округления и восстановления с целью повышения устойчивости к ошибкам и атакам.

Таким образом, проделанная работа закладывает фундамент для дальнейших исследований и развития постквантовых протоколов, способных обеспечить высокую безопасность и эффективность в реальных условиях.

**13. Ссылки и список литературы**

Видео-лекции:

* https://cryptography101.ca/kyber-dilithium/
* <https://ricktube.ru/video?q=https%3A%2F%2Fwww.youtube.com%2Fplaylist%3Flist%3DPLA1qgQLL41SSUOHlq8ADraKKzv47v2yrF>
* <https://www.youtube.com/watch?v=3RdkAQ43eYU>
* https://yandex.ru/video/preview/9278484075604101220

Теория о алгоритмах взлома:

* https://arxiv.org/pdf/2212.12372
* https://ru.wikipedia.org/wiki/Алгоритм\_Шора
* https://ru.wikipedia.org/wiki/Регев,\_Одед
* https://ru.wikipedia.org/wiki/Алгоритм\_Гровера
* https://en.wikipedia.org/wiki/BHT\_algorithm

Русские разработчики и теория:

* https://qapp.tech/help/post-quantum
* https://qapp.tech/help/post-quantum
* https://qapp.tech/help/comparison-pqc-qkd
* https://rqc.ru/
* https://qapp.tech/
* https://qboard.tech/
* https://pki-forum.ru/files/files/2024/19\_09\_24\_Alekseev.pdf?utm\_source=chatgpt.com
* https://ruscrypto.ru/resource/archive/rc2024/files/05\_vysotskaya\_chizhov.pdf
* https://drive.google.com/file/d/1YPtwtpFZitQOs1k2C8NGjogW9YfAWdb1/view

**Документация и теория по Kyber:**

* https://pq-crystals.org/kyber/software.shtml
* https://github.com/pq-crystals/kyber
* <https://cryptopedia.dev/posts/kyber/>

**Приложение 1**

**import random  
import hashlib  
import time  
  
# Параметры для демонстрации (упрощённые и небезопасные)  
#q = 2\*\*8 # Модуль для арифметики в кольце Z\_q  
#p = 2\*\*4 # Новая точность после округления (результат в Z\_p)  
#n = 2\*\*2 # Степень полинома (число коэффициентов)  
#d = q // p # Фактор деления для округления (d = 16)  
#res = []  
  
  
##########################  
# ФУНКЦИИ ДЛЯ ПОЛИНОМОВ  
##########################  
  
def poly\_mul(a, b, mod):  
 *"""Умножение двух полиномов в кольце Z\_mod[x]/(x^n+1)"""* result = [0] \* n  
 for i in range(n):  
 for j in range(n):  
 k = i + j  
 if k < n:  
 result[k] = (result[k] + a[i] \* b[j]) % mod  
 else:  
 result[k - n] = (result[k - n] - a[i] \* b[j]) % mod  
 return [x % mod for x in result]  
  
##########################  
# Кастомизированные функции округления и hint  
##########################  
  
def poly\_round(poly, d, offset):  
 *"""  
 Округление: вычисляем floor((x+offset)/d) для каждого коэффициента.  
 Обычно offset выбирают равным d//2.  
 """* return [int((x + offset) / d) % p for x in poly]  
  
def compute\_hint(poly\_raw, d, threshold):  
 *"""  
 Вычисление hint: для каждого коэффициента, если остаток от деления на d больше или равен threshold, возвращаем 1, иначе 0.  
 """* return [1 if (x % d) >= threshold else 0 for x in poly\_raw]  
  
##########################  
# Кодирование/декодирование сообщения  
##########################  
  
def encode\_message(m):  
 *"""Кодирование: 0 -> 0, 1 -> p//2"""* return [bit \* (p // 2) for bit in m]  
  
def decode\_message(poly):  
 *"""Декодирование: значение >= p//2 интерпретируется как 1, иначе 0"""* return [1 if coeff >= (p // 2) else 0 for coeff in poly]  
  
def serialize\_poly(poly):  
 *"""Сериализация полинома (каждый коэффициент – 1 байт)"""* return bytes(poly)  
  
def hash\_shared(data):  
 *"""Хэширование (SHA-256) для получения общего ключа"""* return hashlib.sha256(data).digest()  
  
##########################  
# Функции для сэмплирования и генерации полиномов  
##########################  
  
def generate\_uniform\_poly(mod):  
 *"""Генерация случайного полинома с коэффициентами из Z\_mod."""* return [random.randrange(mod) for \_ in range(n)]  
  
def sample\_secret():  
 *"""Сэмплирование секретного полинома с малыми коэффициентами (здесь выбираем 0 или 1)."""* return [random.choice([0, 1]) for \_ in range(n)]  
  
##########################  
# Функция reconcile (корректировка)  
##########################  
  
def reconcile(value, hint):  
 if hint == 1 and value < (p - 1):  
 return value + 1  
 elif hint == 0 and value > 0:  
 return value - 1  
 return value  
  
def keygen(offset):  
 a = generate\_uniform\_poly(q)  
 s = sample\_secret() # Секретный ключ для отправителя  
 as\_product = poly\_mul(a, s, q)  
 b = poly\_round(as\_product, d, offset)  
 pk = (a, b) # Публичный ключ  
 # Секретный ключ не передается!  
 return pk, s # Возвращаем только публичный ключ  
  
def encapsulate(pk, offset, threshold):  
 a, b = pk  
 m = [random.choice([0, 1]) for \_ in range(n)]  
 m\_enc = encode\_message(m)  
 s\_prime = sample\_secret() # Эфемерный секрет для сессии  
 u\_product = poly\_mul(a, s\_prime, q)  
 u = poly\_round(u\_product, d, offset)  
 b\_product = poly\_mul(b, s\_prime, q)  
 hint = compute\_hint(b\_product, d, threshold)  
 v\_round = poly\_round(b\_product, d, offset)  
 v = [(v\_round[i] + m\_enc[i]) % p for i in range(n)]  
 ciphertext = (u, v, hint)  
 shared\_key = hash\_shared(serialize\_poly(m)) # Общий ключ для проверки  
 return ciphertext, shared\_key  
  
def decapsulate(ciphertext, sk, offset):  
 u, v, hint = ciphertext  
 us\_product = poly\_mul(u, sk, q)  
 w = poly\_round(us\_product, d, offset)  
 w\_adjusted = [reconcile(w[i], hint[i]) for i in range(n)]  
 m\_enc\_recovered = [(v[i] - w\_adjusted[i]) % p for i in range(n)]  
 m\_recovered = decode\_message(m\_enc\_recovered)  
 shared\_key = hash\_shared(serialize\_poly(m\_recovered))  
 return shared\_key  
  
def test\_kem(trials, offset, threshold):  
 *"""  
 Функция тестирования КЕМ.  
 Возвращает долю успешных итераций.  
 """* success = 0  
 for i in range(trials):  
 pk, sk = keygen(offset)  
 ciphertext, shared\_key\_enc = encapsulate(pk, offset, threshold)  
 shared\_key\_dec = decapsulate(ciphertext, sk, offset)  
 if shared\_key\_enc == shared\_key\_dec:  
 success += 1  
 return success / trials  
  
# Тестирование  
"""q = 2\*\*8 # Модуль для арифметики в кольце Z\_q  
p = 2\*\*4 # Новая точность после округления (результат в Z\_p)  
n = 2\*\*2 # Степень полинома (число коэффициентов)  
d = q // p  
trials = 1  
threshold = 15  
offset = d // 2  
print(test\_kem(trials, offset, threshold))"""  
  
  
start\_time = time.time()  
trials = 10000  
q = 2\*\*12  
p = 32  
n = 8  
threshold = 126  
d = q // p  
offset = d // 2  
print(test\_kem(trials, offset, threshold))  
print(time.time() - start\_time)  
  
  
"""res = []  
  
for q in [2 \*\* 14, 2 \*\* 15]:  
 for p in [32, 64, 128, 256, 512]:  
 for n in [8, 16, 32, 64, 128]:  
 for threshold in range(32, 129):  
 d = q // p  
 if threshold < d:  
 offset = d // 2  
 res.append([p, n, threshold, test\_kem(trials, offset, threshold)])  
  
res.sort(key=lambda x: x[3])  
for i in res:  
 print(\*i)"""  
  
# 2\*\*12  
#256 8 26 0.21  
#64 8 64 0.49  
#32 8 128 0.73  
#32 8 126 0.72**