МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение высшего образования

«Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского» (ННГУ)

Институт информационных технологий, математики и механики

Кафедра: Алгебры, геометрии и дискретной математики

Направление подготовки: «Фундаментальная информатика и информационные технологии»

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Тема:

«Алгоритмы шифрования, основанные на задачах о кратчайшем и ближайшем векторах решетки»

Выполнил: студент группы ___________

_________________________ Напылов Е.И.

Подпись

Научный руководитель:

Доцент,

кандидат физико-математических наук

__________________________ Веселов С.И.

Подпись

Нижний Новгород

2022

Содержание

Введение

В настоящее время вычислительная техника развивается очень быстро. Можно сказать, что мы практически стоим на пороге появления квантовых компьютеров. Этому способствует деятельность таких компаний, как IBM, Google и Intel. Благодаря большому приросту производительности по сравнению с нынешним поколением компьютеров, надежность многих алгоритмов шифрования станет сомнительной. Например, сообщение, закодированное с помощью алгоритма RSA может быть достаточно просто расшифровано с помощью квантового алгоритма Шора.

«Если мы будем ждать слишком долго, будет слишком поздно» - Питер Шор о проблемах современной криптографии. Ученый уже сейчас считает, что внедрение технологий пост-квантовой криптографии с большой вероятностью может не успеть за прогрессом квантовых вычислений. Одними из главных проблем Питер Шор считает «силу воли и время программирования».

Как было сказано ранее, нельзя будет использовать алгоритмы, основанные на разложении числа на простые множители. Потенциальным вариантом решения этой проблемы являются алгоритмы на основе задач решеток. На данный момент самыми известными задачами решеток являются проблемы о поиске кратчайшего и ближайшего векторов решетки. Эти задачи являются сложными с вычислительной точки зрения. Примерами таких алгоритмов являются GGH (основан на задаче ближайшего вектора) и NTRUEncrypt (основан на задаче кратчайшего вектора и кольцах усеченных многочленов).

Как и в большинстве алгоритмов шифрования используются алгоритмически сложные математические задачи. При выборе учитывается главное требование - не должно существовать полиномиальных алгоритмов для их решения. Но не стоит исключать тот факт, что в будущем может появиться быстрый алгоритм решения, подобному тому как появился квантовый алгоритм Шора.

Цель работы.

Целью данной работы является изучение алгоритмов с использованием решеток, а также изучение сложных математических задач решеток, которые обеспечивают их криптографическую стойкость.

Для достижения поставленной цели были поставлены следующие задачи:

1. Изучение принципов и понятий алгоритмов криптографии с открытым ключом.
2. Постановка задачи о ближайшем векторе решетки.
3. Постановка задачи о кратчайшем векторе решетки.
4. Изучение методов решений данных задач.
5. Изучение алгоритмов шифрования GGH и NTRU.
6. Проведение анализа безопасности данных криптосистем с учетом различных видов атак.
7. Реализация алгоритма шифрования NTRU на высокоуровневом языке программирования.

1 Основные понятия и идеи криптографии

Односторонняя функция - основная составляющая современной криптографии. Односторонняя функция - это отображение $F : X \to Y$. Название следует из ее свойства - существует полиномиальный алгоритм отображения $X \to Y$, но при этом не существует полиномиального алгоритма инвертирования $Y \to X$ (вычисление считается крайне трудным).

Это означает, что зная публичный ключ, можно легко зашифровать сообщение, а дешифровать - нет. Быстрая дешифрация возможно только с использованием секретного ключа.

Криптосистемой с открытым ключом (асимметричная система) называется система, имеющая пару ключей - публичный и секретный. Для работы используются 3 основных алгоритма: генерация ключей, шифрование и дешифрование. Данные алгоритмы должны быть полиномиальными, иначе теряется смысл их использования.

Рассмотрим ситуацию с двумя участниками переписки. Пусть собеседник A является получателем зашифрованных сообщений, а B - отправителем. На самом деле в большинстве случаев каждый участник имеет обе описанные роли. Получатель генерирует пару ключей - секретный f и публичный h. Публичный ключ по незащищенному каналу связи отправляется собеседнику B (отправителю). Передача публичного ключа h является безопасной, т.к. он не пригоден для дешифрования сообщения. Отправитель шифрует исходное сообщение с помощью публичного ключа h и отправляет зашифрованное сообщение e собеседнику. Получатель расшифровывает полученное сообщение e с помощью секретного ключа f.

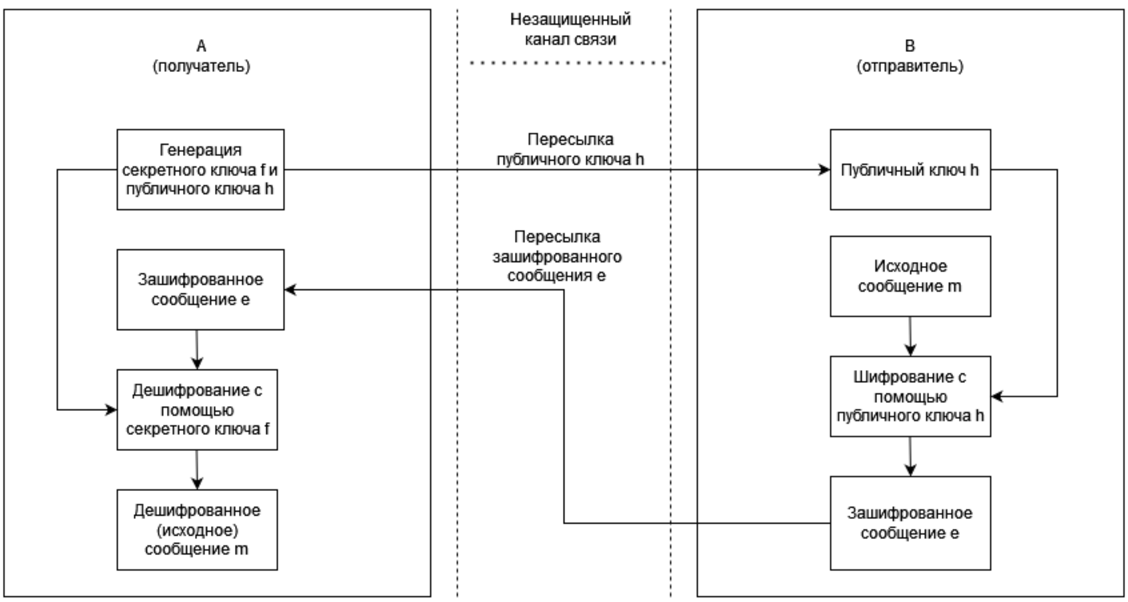

Рисунок 1: Схема криптосистемы с открытым ключом.

2 Задачи решеток, лежащие в основе криптосистем

2.1 Задача о ближайшем векторе решетки

2.1.1 Постановка задачи о ближайшем векторе решетки

Решетка - это подмножество векторного пространства. Пусть $(\vec{b_1},...,\vec{b_n})$ - набор линейно-независимых векторов в евклидовом пространстве $\mathbb{R}^m$, причем $n \leq m$. Решеткой L с базисом $(\vec{b_1},...,\vec{b_n})$ называется множество $L = \{\sum_{i=1}^{n} l_i\vec{b_i} \mid l_i \in \mathbb{Z}\}$. Элементами множества L являются целочисленные линейные комбинации базисных векторов.

Рассмотрим решетку L, заданную в $\mathbb{R}^m$. Пусть задан вектор $\vec{w}$ в $\mathbb{Q}^m$. Ближайшим вектором решетки L для вектора $\vec{w}$ называют такой вектор $\vec{v}$, который удовлетворяет условию: $||\vec{v}-\vec{w}|| \leq ||\vec{x}-\vec{w}||$ для любого вектора $\vec{x}$ из решетки L.

2.1.2 Метод Бабая ближайшей плоскости

Алгоритм Бабая, который будет рассмотрен далее, в общем случае не гарантирует точное решение задачи о ближайшем векторе. Однако, при использовании т.н. хорошего базиса, т.е. полученного в результате работы алгоритма LLL, алгоритм ближайшем плоскости позволяет получить некоторое приближенное решение задачи, которое будет оценено позже.

Пусть решетка L задана базисом $\vec{b_1},...\vec{b_n}$. Пусть $(b_1^*,...,b_n^*)$ - соответствующий базис Грама-Шмидта. Приведем алгоритм, который будет искать ближайший вектор $\vec{v}$ для вектора $\vec{w}$ в решетке L.

Пусть U натянута на систему векторов $(\vec{b_1},...,\vec{b_{n-1}})$ и $L' = L \cap U$ - подрешетка, порожденная базисом $(\vec{b_1},...,\vec{b_{n-1}})$. Рассмотрим плоскость $U+\vec{y}$, $\vec{y} \in L$. Потребуем, чтобы расстояние от исходного вектора $\vec{w}$ этой плоскости было кратчайшим. Следовательно, нам нужно найти такой вектор $\vec{y} \in L$. Найдем вектор $\vec{w'}$, который является ортогональной проекцией вектора $\vec{w}$ на плоскость $U+\vec{y}$. Обозначим разность векторов $\vec{w'}-\vec{y}$ как $\vec{w''}$. Если $\vec{w}$ не входит в решетку, то $\vec{w''}$ так же не входит в эту решетку.

Таким образом мы свели задачу в решетке L для $\vec{w}$ к задаче в решетке L' для вектора $\vec{w''}$. Предположим, что решением полученной задачи будет вектор $\vec{y'}$. Тогда решение исходной задачи получим в виде $\vec{v} = \vec{y}+\vec{y'}$. Заметим, что для полученной вспомогательной задачи можно снова применить описанный метод и так далее.

Лемма 1.

Разложим вектор $\vec{w}$ по ортогональному базису: $\vec{w} = \sum_{j=1}^{n} l_j \vec{b}_j^*, \forall j : l_j \in \mathbb{R}$.

Пусть $\vec{y} = \lfloor l_n \rceil \vec{b}_n \in L$ и $\vec{w}' = \sum_{j=1}^{n-1} l_j \vec{b}_j^* + \lfloor l_n \rceil \vec{b}_n^*$.

Пусть вектор $\vec{y}$ из решетки L представлен в виде $\vec{y} = \lfloor l_n \rceil \vec{b}_n$. Округление до целого числа необходимо для соблюдения целочисленности. Представим вектор $\vec{w}'$ в виде $\vec{w}' = \sum_{j=1}^{n-1} l_j \vec{b}_j^* + \lfloor l_n \rceil \vec{b}_n^*$.

Тогда расстояние от плоскости $U + \vec{y}$ до $\vec{w}$ будет наименьшим. При этом $\vec{w}'$ является ортогональной проекцией вектора $\vec{w}$ на плоскость $U + \vec{y}$

Доказательство:

Расстояние между $\vec{w}$ и $U + \vec{y}$ является нижней границей множества всех векторов, удовлетворяющих неравенству для любого вектора $\vec{u}$ из U: $||\vec{w} - (\vec{u} + \vec{y})||$.

Пусть $\vec{y} = \sum_{j=1}^{n} l_j' \vec{b}_j$ - произвольный элемент L для любого $l_j' \in \mathbb{Z}$ Тогда вектор $\vec{y}$ может быть представлен в виде: $\vec{y} = \sum_{j=1}^{n-1} l_j'' \vec{b}_j^* + l_n' \vec{b}_n^*$ для любого $l'' \in \mathbb{R}, j \in [1..n-1]$.

Рассмотрим расстояние как функцию: $f(\vec{u}) = ||\vec{w} - (\vec{u} + \vec{y})||$. Минимум данной функции достигается при $\vec{u} = \sum_{j=1}^{n-1} (l_j - l_j'') \vec{b}_j^*$

Из этого следует, что $l' = \lfloor l_n \rceil$, и поэтому выбор $\vec{y}$ в лемме является верным. Из этого следует, что коэффициент l' необходимо округлить до ближайшего целого числа. Другими словами, при $\vec{y} = \lfloor l_n \rceil \vec{b}_n$ действительно достигается наименьшее расстояние между $\vec{w}$ и $U + \vec{y}$

Переходим к доказательству второй части.

Вектор $\vec{w}'$ удовлетворяет условию: $\vec{w}' = \sum_{j=1}^{n-1} l_j \vec{b}_j^* + \lfloor l_n \rceil (b_n^* - \vec{b}_n) \in U$.

Отсюда следует, что вектор $\vec{w}'$ лежит на $U + \vec{y}$

Рассмотрим вектор $\vec{d} = \vec{w} - \vec{w}' = \sum_{j=1}^{n} l_j \vec{b}_j^* - \sum_{j=1}^{n-1} l_j \vec{b}_j^* - \lfloor l_n \rceil \vec{b}_n^* = (l_n - \lfloor l_n \rceil) \vec{b}_n^*$. Этот вектор ортогонален U. Отсюда следует, что $\vec{w}'$ лежащий в $U + \vec{y}$ является ортогональной проекцией $\vec{w}$ на $U + \vec{y}$.

Лемма 2.

Рассмотрим базис $(\vec{b}_1, ..., \vec{b}_n)$, который был редуцирован по LLL алгоритму ($\sigma = 3/4$). Пусть $\vec{v}$ получен методом ближайшей плоскости для вектора $\vec{w}$. Тогда:

$||\vec{w} - \vec{v}||^2 \leq \frac{(2^n - 1)}{4} ||\vec{b}_n^*||$.

Доказательство:

Доказательство будем проводить по индукции. Предположим, что утверждение верно при $n - 1$. Докажем, что оно верно и при n.

Исходя из предположения $||\vec{w}'' - \vec{y}'||^2 \leq \frac{2^{n-1} - 1}{4} ||b_{n-1}^*||^2$, где $\vec{y}' = \vec{v} - \vec{y}, \vec{y} \in L, \vec{w}'' = \vec{w}' - \vec{y}$

$||\vec{w} - \vec{v}|| = ||\vec{w} - (\vec{y} + \vec{y}')||^2 = ||\vec{w} - \vec{w}' + \vec{w}' - (\vec{y} + \vec{y}')||^2 =$

$= ||\vec{w} - \vec{w}'||^2 + ||\vec{w}'' - \vec{y}'||^2 \leq \frac{||\vec{b}_n^*||^2}{4} + \frac{(2^{n-1} - 1)||\vec{b}_{n-1}^*||^2}{4} \leq (\frac{1}{4} + 2\frac{2^{n-1} - 1}{4})||\vec{b}_n^*||^2 = \frac{(2^n - 1)}{4} ||\vec{b}_n^*||$

Теорема 1.

Пусть базис $(\vec{b_1},...,\vec{b_n})$ получен с помощью алгоритма LLL с коэффициентом σ = 3/4. Пусть вектор $\vec{v}$ был получен в результате работы алгоритма Бабая для $\vec{w}$. Тогда расстояние между этими векторами можно оценить сверху как:

$||\vec{v}-\vec{w}|| < 2^{n/2}||\vec{u}-\vec{w}||, \forall\vec{u} \in L.$

Доказательство:

Доказательство будем проводить по индукции. Предположим, что утверждение верно при $n-1$. Докажем, что оно верно и при n.

Пусть $\vec{u} \in L$ - ближайший вектор для $\vec{w}$, а $\vec{y}$ был выбран на первом шаге алгоритма Бабая. Рассмотрим 2 случая:

1. Если $\vec{u} \in U+\vec{y}$.

Тогда $||\vec{u}-\vec{w}||^2 = ||\vec{u}-\vec{w'}||^2 + ||\vec{w'}-\vec{w}||^2$, поэтому $\vec{u}$ - ближайший вектор для $\vec{w'}$. Следовательно, $\vec{u}-\vec{y}$ - ближайший вектор для $\vec{w''} = \vec{w'}-\vec{y} \in U$.

Пусть $\vec{y'}$ получен в результате алгоритма Бабая для $\vec{w''}$.

По предположению индукции $||\vec{y'}-\vec{w''}|| < 2^{(n-1)/2}||\vec{u}-\vec{y}-\vec{w''}||$.

Подставим $\vec{w'}-\vec{y}$ вместо $\vec{w''}$: $||\vec{y}+\vec{y'}-\vec{w'}|| < 2^{(n-1)/2}||\vec{u}-\vec{w'}||$.

Получили $||\vec{v}-\vec{w}||^2 = ||\vec{y}+\vec{y'}-\vec{w'}||^2 = ||\vec{w'}-\vec{w}||^2 < 2^{(n-1)}||\vec{u}-\vec{w'}||^2 + ||\vec{w'}-\vec{w}||^2$.

Используя $||\vec{u}-\vec{w'}||, ||\vec{w'}-\vec{w}|| \leq ||\vec{u}-\vec{w}||, 2^{n-1}+1 \leq 2^n$ получили результат.

2. Если $\vec{u} \notin U+\vec{y}$.

Как только расстояние от $\vec{w}$ до $U+\vec{y}$ будет $\leq \frac{||\vec{b_n^*}||}{2}$, будет выполнятся неравенство $||\vec{w}-\vec{u}|| \geq \frac{||\vec{b_n^*}||}{2}$.

Используя предыдущую лемму получим: $\frac{||\vec{b_n^*}||}{2} \geq \frac{1}{2}\sqrt{\frac{4}{2^n-1}}||\vec{w}-\vec{v}||$. Отсюда следует, что

$||\vec{w}-\vec{v}|| \leq 2^{n/2}||\vec{w}-\vec{u}||$

2.1.3 Метод округления Бабая

Рассмотрим еще один метод. Идея заключается в вычислении коэффициентов разложения исходного вектора по базису решетки. В общем случае полученные коэффициенты не будут являться целыми числами. Поэтому их нужно округлить до ближайшего целого. Решением задачи ближайшего вектора будет линейная комбинация базисных векторов решетки с вычисленными коэффициентами. Метод округления имеет гораздо меньшую вычислительную сложность за счет отсутствия процесса ортогонализации. Но при этом данный алгоритм все так же не гарантирует точное решение задачи. Точность метода будет оценена позже.

Рассмотрим решетку L, заданную базисом $(\vec{b_1},...,\vec{b_n})$. Требуется найти ближайший вектор для вектора $\vec{w}$.

Разложим $\vec{w}$ по базису решетки: $\vec{w} = \sum\limits_{i=1}^{n} \alpha_i \vec{b_i}$. Чтобы найти α_i достаточно решить систему уравнений $(\vec{b_1},...,\vec{b_n})\vec{\alpha} = \vec{w}$ На данном этапе коэффициенты разложения - действительные числа. Теперь округлим коэффициенты до ближайших целых чисел:
$(\alpha_i,...,\alpha_n) \to (\lfloor\alpha_i\rceil,...,\lfloor\alpha_i\rceil)$. Ближайший вектор может быть разложен по базису благодаря округлению коэффициентов $\vec{v} = \sum\limits_{i=1}^{n} \lfloor\alpha_i\rceil \vec{b_i}$. Нетрудно заметить, что из-за округления векторы будут удовлетворять условию:
$\vec{w} - \vec{v} = \sum\limits_{i=1}^{n} \gamma_i \vec{b_i}$,где $|\gamma_i| \leq \frac{1}{2}$.

Теорема 2.

Рассмотрим решетку L, заданную базисом $(\vec{b_1},...,\vec{b_n})$, полученным с помощью алгоритма LLL с коэффициентом σ = 3/4. Тогда вектор $\vec{v}$, полученный в результате метода округления для $\vec{w}$ будет удовлетворять неравенству $||\vec{w} - \vec{v}|| \leq (1 + 2n(\frac{9}{2})^{n/2})||\vec{w} - \vec{u}||, \forall \vec{u} \in L$.

2.1.4 Метод вложения

Рассмотрим решетку L, заданную базисом $(\vec{b_1},...,\vec{b_n})$. Пусть задан вектор $\vec{w}$, для которого нужно найти ближайший вектор. Данный вектор можно представить в приближенном виде $\vec{w} \approx \vec{v} = \sum\limits_{i=1}^{n} l_i \vec{b_i}$.

Очевидно, что расстояние между векторами $\vec{w}$ и $\vec{e}$ должно быть минимальным. Пусть разницей между ними будет $\vec{e} = \vec{w} - \sum\limits_{i=1}^{n} l_i \vec{b_i}$. Идея заключается в том, чтобы определить решетку L', в которой находится кратчайший вектор $\vec{e}$.

Для того, чтобы определить базис решетки L' возьмем действительное положительное число M. Дополним базисные векторы новой компонентой: $(([\vec{b_1},0],..,[\vec{b_n},0])$ и $[\vec{w},M]$ Таким образом мы перешли к задаче о кратчайшем векторе решетки L', заданной базисом $([\vec{b_1},0],..,[\vec{w},M])$.

Осталось найти этот кратчайший вектор $\vec{e}$ и вычислить ближайший вектор как $\vec{w} - \vec{e}$. Решить задачу о кратчайшем векторе можно, например, использовав алгоритм LLL. Подробнее алгоритм Ленстры-Ленстры-Ловаса рассмотрен в пункте 2.2.3.

Лемма 3.

Пусть $(\vec{b_1},...,\vec{b_n})$ - базис решетки целочисленной решетки L, а λ_1 - длина кратчайшего вектора решетки L. Пусть $\vec{v}$ - ближайший вектор решетки для вектора $\vec{w}$. Разницу между этими векторами обозначим как $\vec{e} = \vec{w} - \vec{v}$. Предположим, что $||\vec{e}|| < \lambda_1/2$ и $M = ||\vec{e}||$. Обозначим дополненный компонентой M вектор $\vec{e}$ как $(\vec{e},M)$. Тогда вектор $(\vec{e},M)$ является кратчайшим вектором решетки L'.

Доказательство:

Любой вектор решетки L' может быть представлен в виде $l_{n+1}(\vec{e},M) + \sum\limits_{i=1}^{n} l_i(\vec{b_i},0), \forall (l_1,...,l_{n+1}) \in \mathbb{Z}$. Каждый ненулевой вектор, для которого $l_{n+1} = 0$ имеет длину не меньше, чем λ_1. Из равенства $||(\vec{e},M)||^2 = ||\vec{e}||^2 + M^2 = 2M^2 < 2\lambda_1^2/4$ следует, что $||(\vec{e},\pm M)|| > \frac{\lambda_1}{\sqrt{2}}$. Так как $\vec{v}$ - ближайший вектор до $\vec{w}$, то $||\vec{e}|| \leq ||\vec{e}+\vec{x}||, \forall \vec{x} \in L$, а так же $(\vec{u},M) \in L'$ имеет как минимум ту же длину. Теперь предположим, что $|l_{n+1}| \geq 2$, тогда $||(\vec{u},l_{n+1}M)||^2 \geq ||(0,l_{n+1}M)||^2 \geq (2M)^2$, следовательно, $||(\vec{u},l_{n+1}M)|| \geq 2||(\vec{e},M)||$.

Из этого следует, что вектор $(\vec{e},M)$ имеет наименьшую длину среди всех векторов решетки L'.

2.2 Задача о кратчайшем векторе решетки

2.2.1 Постановка задачи о кратчайшем векторе решетки

Ненулевой вектор, принадлежащий решетке и имеющий наименьшую длину, называется кратчайшим вектором данной решетки. Пусть решетка L задана базисом $(\vec{b}_1, ..., \vec{b}_n)$. Требуется найти такой вектор $\vec{v} \in L$, для которого $||\vec{v}|| = min||\vec{v}_i||, \forall \vec{v}_i \in L$

Если базис решетки является ортогональным и базисные векторы относительно короткие, то задача поиска кратчайшего вектора решается достаточно просто. Но что делать если базис не является ортогональным? Первое, что может прийти в голову - применить процесс ортогонализации Грама-Шмидта - алгоритм приведения базиса к ортогональному. Но, в рамках задач решеток в множестве $\mathbb{Z}$, сделать это не получится. Этот процесс в общем случае не гарантирует целочисленность.

Оказывается, что решить такую задачу можно с помощью приведения исходного базиса к базису, который как можно ближе к ортогональному и как можно короче.

2.2.2 Алгоритм Лагранжа-Гаусса редукции базиса для двумерного случая

Упорядоченный базис $(\vec{b_1}, \vec{b_2})$ является редуцированным по Лагранжу-Гауссу, если $||\vec{b_1}|| \leq ||\vec{b_2}|| \leq ||\vec{b_2} + q\vec{b_1}||, \forall q \in \mathbb{Z}$.

Теорема 3.

Пусть λ_1, λ_2 - последовательные минимумы для решетки L, т.е. $\lambda_1 < \lambda_2$ - длины кратчайших векторов решетки. Если $(\vec{b_1}, \vec{b_2})$ - базис, редуцированный по Лагранжу-Гауссу, то $||\vec{b_1}|| = \lambda_1$, $||\vec{b_2}|| = \lambda_2$

Доказательство:

По определению базиса Лагранжа-Гаусса: $||\vec{b_2} + q\vec{b_1}|| \geq ||\vec{b_2}|| \geq ||\vec{b_1}||$.
Пусть $\vec{v} = l_1\vec{b_1} + l_2\vec{b_2}$ и $||\vec{v}|| \neq 0$. Если $l_2 = 0$, то $||\vec{v}|| \geq ||\vec{b_1}||$.
В противном случае $l_1 = ql_2 + r, q \in \mathbb{Z}, r \in \mathbb{Z}$ и $0 \leq r \leq |l_2|$. Тогда $\vec{v} = r\vec{b_1} + l_2(\vec{b_2} + q\vec{b_1})$.
Применяя неравенству треугольника получим: $\vec{v} \geq |l_2|||\vec{b_2} + q\vec{b_1}|| - r||\vec{b_1}|| = (|l_2| - r)||\vec{b_2} + q\vec{b_1}|| + r(||\vec{b_2} + q\vec{b_1}|| - ||\vec{b_1}||) \geq ||\vec{b_2} + q\vec{b_1}|| \geq ||\vec{b_2}|| \geq ||\vec{b_1}||$. Т.е. $||\vec{v}|| \geq ||\vec{b_2}|| \geq ||\vec{b_1}||$. Это и означает то, что $(\vec{b_1}, \vec{b_2})$ являются самыми короткими векторами решетки, т.е. $||\vec{b_1}|| = \lambda_1$, $||\vec{b_2}|| = \lambda_2$.

Идея алгоритма Лагранжа-Гаусса заключается в том, что расстояние между векторами $\vec{b_2}$ и $\vec{b_1}\mu$ становится минимальным при $\mu = \frac{(\vec{b_1}, \vec{b_2})}{(\vec{b_1}, \vec{b_1})}$. Число μ является коэффициентом ортогонализации из метода Грама-Шмидта. Так как набор $(\vec{b_1}, \vec{b_2})$ является базисом решетки, а не пространства, то $\vec{b_2}$ можно заменить на $\vec{b_2} - \lfloor \mu \rceil \vec{b_1}$. Таким образом на каждой итерации алгоритма вектор $\vec{b_2}$ будет редуцироваться, а вектор $\vec{b_1}$ будет заменяться на предыдущее состояние вектора $\vec{b_2}$. Итерации необходимо проводить до тех пор, пока $||\vec{b_1}|| \geq ||\vec{b_2}||$.

Лемма 4.

Пусть $(\vec{b_1}, \vec{b_2})$ - упорядоченный базис решетки L. Этот базис является редуцированным по Лагранжу-Гауссу тогда и только тогда, когда $||\vec{b_1}|| \leq ||\vec{b_2}|| \leq ||\vec{b_2} \pm \vec{b_1}||$.

Доказательство:

Доказательство слева направо следует из того, что при $q = 1$ из условия Лагранжа-Гаусса получается условие из леммы. В обратную сторону доказательство следует из того, что минимум функции $||\vec{b_2} + \alpha\vec{b_1}||^2$ достигается при $|\alpha| < 1$, следовательно, при $q > 1$ и при $q < -1$ выполняется условие Лагранжа-Гаусса.

2.2.3 Алгоритм LLL редукции базиса

Пусть набор векторов $(\vec{b_1},...,\vec{b_n})$ является базисом решетки L. Приведем этот базис к ортогональному виду с помощью процесса ортогонализации Грама-Шмидта. В результате применения алгоритма получим линейно независимый набор векторов $(\vec{b_1^*},...,\vec{b_n^*})$. Замечание: данный набор векторов в общем случае не может являться новым базисом решетки L. Кроме ортогонального набора векторов результатом алгоритма является набор чисел $\mu_{i,j}$ - набор коэффициентов ортогонализации. Согласно алгоритму Грама-Шмидта коэффициенты имеют вид:
$\mu_{i,j} = \frac{(\vec{b_i},\vec{b_j^*})}{(\vec{b_j^*},\vec{b_j^*})}$.

Упорядоченный базис $(\vec{b_1},...,\vec{b_n})$ является LLL-редуцированным (или LLL-приведенным) с параметром σ, если выполняются следующие условия:

1. $|\mu_{ij}| \leq \frac{1}{2}, \forall i,j : 1 \leq j \leq i \leq n$
2. $(\vec{b_i^*},\vec{b_i^*}) \geq (\sigma - \mu_{i,i-1}^2)(\vec{b_{i-1}^*},\vec{b_{i-1}^*})\ \forall i : 2 \leq i \leq n$

где $\sigma : \frac{1}{4} \leq \sigma \leq 1$. Чаще всего в качестве параметра используют $\sigma = \frac{3}{4}$. Первое условие показывает редукцию размера, второе условие называется условием Ловаса.

Лемма 5.

Пусть $(\vec{b_1},...,\vec{b_n})$ - базис решетки $L \in \mathbb{R}^m$, а $(\vec{b_1^*},...,\vec{b_n^*})$ - соответствующий базис Грама-Шмидта. Пусть кратчайший вектор данной решетки $\vec{v}$ имеет длину $||\vec{v}||$, тогда
$||\vec{v}|| \geq min(||\vec{b_1^*}||,...,||\vec{b_n^*}||)$.

Алгоритм LLL (Ленстры-Ленстры-Ловаса) редукции базиса решетки

На вход алгоритма поступает базис решетки. Над базисом проводится процесс ортогонализации Грама-Шмидта, при этом сохраняются коэффициенты ортогонализации $\mu_{i,j}$. Для того, чтобы получить LLL-редуцированный базис, необходимо на каждой итерации выбирать целочисленные линейные комбинации. Если окажется, что условие Ловаса не выполняется, то это означает, что текущий базисный вектор совсем немного длиннее предыдущего. В этом случае необходимо произвести перестановку этих векторов и пересчитать ортогонализацию.

Ниже приведен полный алгоритм Ленстры-Ленстры-Ловаса.

1. Вычисляется ортогональный набор векторов $(\vec{b}_1^*, ..., \vec{b}_n^*)$ и сохраняется набор коэффициентов Грама-Шмидта $\mu_{ij}, \forall i, j$: $1 \leq j \leq i \leq n$

2. Вычисляются длины векторов из полученного набора: $B_i = ||\vec{b}_i^*||^2, \forall i \in [1..n]$

3. $k = 2$

4. Пока $k \leq n$:

   5.1. Для $j = k-1, ..., 1$:

      5.1.1. $q_j = \lfloor \mu_{kj} \rceil$

      5.1.2. $\vec{b}_k = \vec{b}_k - q_j \vec{b}_j$

      5.1.3. Пересчитываются коэффициенты $\mu_{kj}, \forall j \in [1..k)$

   5.2. Если $B_k \geq (\sigma - \mu_{k,k-1}^2) B_{k-1}$, то $k = k+1$

   5.3. Если $B_k < (\sigma - \mu_{k,k-1}^2) B_{k-1}$, то:

      5.3.1. Перестановка $\vec{b}_k$ и $\vec{b_{k-1}}$

      5.3.2. Для $j = 1, ..., k-1$

         5.3.2.1. Пересчитываются значения $\vec{b}_k^*, \vec{b_{k-1}^*}$

         5.3.2.2. Пересчитываются длины B_k, B_{k-1}

         5.3.2.3. Пересчитываются коэффициенты $\mu_{k-1,j}$

         5.3.2.4. Пересчитаются коэффициенты $\mu_{k,j}$

      5.3.3. Для $i = k+1, ..., n$:

         5.3.3.1. Пересчитываются коэффициенты $\mu_{i,k}$

         5.3.3.2. Пересчитываются коэффициенты $\mu_{i,k-1}$

      5.3.4. $k = max\{2, k-1\}$

Базис, полученный в результате работы алгоритма LLL, позволяет получить некоторое приближение для решения задачи о кратчайшем векторе решетки.

Для оценки данного приближения потребуется лемма.

Лемма 6. Пусть $(\vec{b}_1,..,\vec{b}_n)$ является базисом решетки $L \subset \mathbb{R}^m$, полученным в результате работы LLL алгоритма с коэффициентом $\sigma = \frac{3}{4}$, а $(\vec{b}_1^*,..,\vec{b}_n^*)$ - базис Грама-Шмидта, тогда:

1. $||\vec{b}_j^*||^2 \leq 2^{i-j}||\vec{b}_i^*||^2, \forall 1 \leq j \leq i \leq n$
2. $||\vec{b}_j^*||^2 \leq ||\vec{b}_i||^2 \leq (1/2+2^{i-2})||\vec{b}_i^*||^2, \forall 1 \leq i \leq n$
3. $||\vec{b}_j|| \leq 2^{(i-1)/2}||\vec{b}_i^*||, \forall 1 \leq j \leq i \leq n$

Доказательство

1. Из условия Ловаса (из определения LLL базиса) следует, что $||\vec{b}_i^*||^2 \geq (1/2)||\vec{b}_{i-1}^*||^2$. По индукции получим результат: $||\vec{b}_j^*||^2 \leq 2^{i-j}||\vec{b}_i^*||^2, \forall 1 \leq j \leq i \leq n$

2. Из того, что $\vec{b}_i = \vec{b}_i^* + \sum\limits_{j=1}^{i-1} \mu_{i,j}\vec{b}_j^*$ следует, что

$||\vec{b}_i||^2 = (\vec{b}_j^* + \sum\limits_{j=1}^{i-1} \mu_{i,j}\vec{b}_j^*, \vec{b}_j^* + \sum\limits_{j=1}^{i-1} \mu_{i,j}\vec{b}_j^*) = ||\vec{b}_i^*||^2 + \sum\limits_{j=1}^{i-1} \mu_{i,j}^2||\vec{b}_j^*||^2 \geq ||\vec{b}_i^{*2}||$. Из предыдущего пункта $||\vec{b}_i^*||^2(1+1/4)\sum\limits_{j=1}^{i-1} 2^{i-j} = ||\vec{b}_i^*||^2(1/2+2^{i-2})$

3. Начиная с $j \geq 1$ будет выполняться неравенство: $1/2+2^{j-2} \leq 2^{j-1}$. В таком случае, пользуясь предыдущим пунктом, получим: $||\vec{b}_j||^2 \leq 2^{j-1}||\vec{b}_j^*||^2$. Используя первый пункт: $||\vec{b}_j||^2 \leq 2^{j-1}2^{i-j}||\vec{b}_i^*||^2 = 2^{i-1}||\vec{b}_i^*||^2$. И наконец получаем: $||\vec{b}_j|| \leq 2^{(i-1)/2}||\vec{b}_i^*||$

Теорема 4. Пусть $(\vec{b}_1,..,\vec{b}_n)$ является базисом решетки $L \subset \mathbb{R}^m$, полученным в результате работы LLL алгоритма с коэффициентом $\sigma = \frac{3}{4}$ и пусть $\vec{v}$ - кратчайший вектор решетки L, тогда $||\vec{b}_1|| \leq 2^{(n-1)/2}||\vec{v}||$

Доказательство:

Из первого пункта леммы 6 при $i = 1$ следует, что $||\vec{b}_i^*|| \geq 2^{(i-1)/2}||\vec{b}_1^*||$. Следовательно, $||\vec{v}|| \geq min(||\vec{b}_1^*||,...,||\vec{b}_n^*||) \geq min(2^{(i-1)/2}||\vec{b}_1^*||) = 2^{(1-n)/2}||\vec{b}_1^*||$. Из того, что процесс ортогонализации Грама-Шмидта не затрагивает первый вектор следует: $||\vec{b}_1|| \leq 2^{(n-1)/2}||\vec{v}||$.

Таким образом, кратчайший вектор решетки может быть приблизительно найден как первый вектор из LLL-редуцированного базиса ($\sigma = 3/4$), при этом его длина будет не более чем в $2^{(n-1)/2}$ раз больше. Несмотря на то, что алгоритм не позволяет найти точное решение задачи, он может быть полезным для сужения области поиска при своей относительно невысокой сложности при $\sigma \in (\frac{1}{4}...1)$.

3 Схема шифрования GGH

Схема шифрования GGH была разработана в 1997 году. Свое название система получила благодаря ученым, которые ее разработали: Одд Голдрейх, Гольдвассер Шафи, Шай Халеви. GGH - схема шифрования с открытым ключом, основанная на задачах о кратчайшем и ближайшем векторах решетки.

3.1 Создание пары ключей

Секретным ключом схемы шифрования GGH является матрица B, состоящая из базисных векторов решетки L. Секретный ключ должен обладать свойствами так называемого хорошего базиса. Хороший базис решетки - базис, который состоит из коротких векторов и при этом векторы близки к ортогональным.

Публичным ключом схемы является матрица B’, состоящая из некоторых других базисных векторов решетки L. Базис B’ не должен обладать свойствами хорошего базиса. Матрица B’ может быть получена из матрицы B с помощью некоторых действий. Даниэль Миксио предложил использовать Эрмитову нормальную форму матрицы B. Существует другой способ получения B’ - необходимо ввести секретный параметр T - унимодулярную матрицу, тогда $B' = TB$, $det(T) = 1$.

3.2 Шифрование

Сообщение, которое необходимо зашифровать должно быть преобразовано в представление в виде целых чисел. Обозначим его $m \in Z^n$. Введем короткий вектор ошибки $e = (\delta_1\sigma, ..., \delta_n\sigma)$, где $\delta_i = -1$ или $\delta_i = 1$, а σ - любое небольшое число. Пусть c - зашифрованное сообщение. Оно может быть получено по формуле: $c = mB' + e$, где e - вектор ошибки.

3.3 Дешифрование

Для расшифровки используется секретный ключ B - ”хороший” базис. Производится вычисление: $c' = cB^{-1} = mT + eB^{-1}$. В векторе c' по прежнему содержится короткий вектор ошибки. Базис B позволяет легко вычислить кратчайший вектор решетки, тем самым откинув вектор ошибки. Например, может быть использован метод округления Бабая, в итоге: $m = mTT^{-1}$.

3.4 Атака с использованием поиска ближайшего вектора

Идея заключается в поиске ближайшего вектора в публичном ключе, т.е. в ”плохом” базисе. Для данной процедуры могут использоваться алгоритмы ближайшей плоскости и метода округления Бабая, алгоритм вложения. Подобные атаки эффективны только при небольших размерах ключа.

3.5 Атака с использованием редукции публичного ключа

Идея заключается в поиске ”хорошего” базиса, в котором возможен эффективный поиск ближайших векторов. Для получения такого базиса необходимо редуцировать открытый ключ. Процедура редукции может быть проведена с использованием алгоритма LLL. Использование данного подхода эффективно только при небольшой размерности решетки ($N < 100$).

Таким образом, для достижения высокой безопасности требуется большая размерность решетки, а из того, что базис зависит квадратично от размерности решетки следует большой расход памяти и нагрузка на канал связи.

4 Схема шифрования NTRUEncrypt

4.1 Основы

NTRUEncrypt - система шифрования с открытым ключом. На момент появления система называлась просто NTRU. Данная система шифрования была разработана Джеффри Хоффштейном, Джиллом Пайпером и Джозефом Сильверманом в 1996 году. NTRU считается первой криптосистемой, которая может быть применена для решения настоящих практических задач. Данная криптографическая система создавалась специально в качестве конкурента самой популярной на данной момент RSA, постквантовую стойкость которой в 1994 году разрушил алгоритм Питера Шора.

Система NTRUEncrypt основана на трудной вычислительной задаче о поиске кратчайшего вектора решетки и кольцах усеченных многочленов.

Рассмотрим множество S в котором определены операции сложения + и умножения *. Пусть в этом множестве:

1. Операция сложения является ассоциативной и коммутативной
2. Для сложения определен нулевой элемент
3. Для операции сложения существуют обратные элементы
4. Определено свойство дистрибутивности

Тогда $R = (S, +, *)$ является кольцом.

Рассмотрим множество многочленов над кольцом R, в котором определены операции сложения и умножения. В таком случае Это множество $R[x]$ так же является кольцом. Такое кольцо называется кольцом многочленов.

Доказательство:

R является абелевой группой, т.к. для многочленов выполняются свойства коммутативности и ассоциативности, определен нулевой многочлен, существует обратный относительно сложения. Для многочленов выполняется свойство дистрибутивности.

Сложение в кольце многочленов выполняется по правилу:

$A + B = \sum\limits_{0}^{\infty} a_i x^i + \sum\limits_{0}^{\infty} b_i x^i = \sum\limits_{0}^{\infty} (a_i + b_i) x^i$

Умножение в кольце многочленов выполняется по правилу:

$A * B = \sum\limits_{0}^{\infty} a_i x^i * \sum\limits_{0}^{\infty} b_i x^i = \sum\limits_{0}^{\infty} (\sum\limits_{j=0}^{i} a_j b_{i-j}) x^i$

В NTRU используется кольцо усеченных многочленов $R = Z[x]/(x^N - 1)$. Степень многочленов не превосходит $N-1$. Операция сложения в таком кольце ничем не отличается от сложения в обычном кольце многочленов. С операцией умножения дело обстоит немного сложнее. Дело в том, что при обычном умножении многочленов степень результата скалывается из степеней множителей. В случае усеченных многочленов умножение происходит по правилу:

1. Если при умножении получается слагаемое с x^α, где $\alpha \leq N-1$, то оно не изменяется.

2. Если при умножении получается слагаемое с x^α, где $\alpha > N-1$, то данное слагаемое заменяется по правилу: $x^N \to 1$, $x^{N+1} \to x$, $x^{N+2} \to x^2$ и т.д.

Правило умножения может быть записано в формульном представлении:

$$C = A * B;$$

$$\sum_{0}^{N-1} c_i x^i = \sum_{0}^{N-1} a_i x^i * \sum_{0}^{N-1} b_i x^i, \text{ где } c_k = \sum_{i=0}^{k} a_i b_{k-i} + \sum_{i=k+1}^{N-1} a_i b_{N+k-i}.$$

4.2 Параметры криптосистемы NTRU

Криптосистема NTRUEncrypt использует следующие параметры:

1. N - целое число, определяющее степень многочленов

2. p и q - целые числа, определяющие модули, по которым будут выполнятся операции сложения и умножения. Числа должны удовлетворять условиям: НОД$(p,q) = 1$, $p << q$.

3. M_f, M_g, M_ϕ, M_m - различные множества многочленов степени $N-1$, имеющих целые коэффициенты.

4.3 Создание пары ключей

Для создания ключей клиент, который будет принимать зашифрованные сообщения генерирует случайные многочлены $f(x), g(x) \in M_g$.

Для многочлена $f(x)$ должны существовать обратные элементы как для умножения по модулю p, так и для для умножения по модулю p. Если полученный многочлен не удовлетворяет этому условию, то в таком случае выбирается новая пара многочленов f, g.

Коэффициенты многочленов f и g - элементы множества {-1, 0, 1}. И f и g имеют t коэффициентов равных единице. Многочлен f имеет $t-1$ коэффициентов равных минус единице. Многочлен g имеет t минус единиц. Данные многочлены можно считать небольшими.

Обратные многочлены обозначим как f_p и f_q. По определению обратных элементов должны выполняться условия (по модулям):
$f_p * f = 1 \ (mod\ p)$ и $f_q * f = 1 \ (mod\ q)$

Секретным ключом является пара многочленов - f и обратный элемент f_p.

Публичный ключ по сравнению с секретным должен быть большим. Публичный ключ обозначим как h.

$h = f_q * g \ (mod\ q)$.

Публичный ключ передается по незащищенному каналу связи другому клиенту, который собирается отправлять зашифрованные сообщения. Данный шаг является безопасным, т.к. публичный ключ является односторонним, т.е. позволяет только зашифровать сообщение.

Таким образом, в результате этапа генерации ключей имеем:

Секретный ключ: пара многочленов f и f_p.

Публичный ключ: многочлен $h = f_q * g \ (mod\ q)$.

4.4 Шифрование

Для шифрования необходимо исходное сообщение представить в виде многочлена из множества M_m. Для этого сообщение представляется в троичном виде с алфавитом {-1, 0, 1 }. Т.е. для кодирования сообщения нет никаких ограничений на распределение вероятностей коэффициентов. Обозначим этот многочлен как m.

Далее выбирается случайный многочлен ϕ из множества M_ϕ. Коэффициенты многочлена ϕ являются элементами множества {-1, 0, 1}. Зачастую вероятность выпадения нулевого коэффициента больше, чем у остальных коэффициентов.

После этого сообщение m шифруется с помощью публичного ключа h, полученного от клиента, который собирается принимать зашифрованные сообщения. Операции выполняются по модулю q. Обозначим зашифрованное сообщение как e:

$e = p\phi * h + m \ (mod \ q)$

Умножение случайного ϕ на h дает большой многочлен. Многочлен m добавляет совсем небольшое изменение.

4.5 Дешифрование

Для дешифровки полученного сообщения e используется секретный ключ f и его обратный элемент - f_p.

Вычисляется многочлен $a = f * e \ (mod \ q)$, в котором каждый коэффициент α_i выбирается таким, чтобы $\frac{-q}{2} \leq \alpha_i \leq \frac{q}{2}$. Данная процедура называется центрированием.

Полученный многочлен a рассматривается как многочлен с целочисленными коэффициентами. Исходное сообщение получается по формуле:

$m = f_p * a \ (mod \ p)$.

4.6 Корректность дешифрования

Необходимо удостовериться в том, что после применения алгоритмов шифрования и дешифрования сообщение может быть получено в исходном виде. Пусть сообщение m было зашифровано в сообщение $e = p\phi * h + m \ (mod\ q)$. Дешифрование начинается с вычисления многочлена $a = f * e$. Подставим в это выражение формулу, с помощью которой сообщение было зашифровано:

$a = f(p * \phi * h + m) \ (mod\ q) = p\phi * f * h + f * m \ (mod\ q)$

Воспользуемся тем, что публичный ключ h - это произведение инверсии f_q на g.

$a = p\phi * f * h + f * m \ (mod\ q) = p\phi * f * (f_q * g) + f * m \ (mod\ q)$

$a = p\phi * f * f_q * g + f * m \ (mod\ q)$

В полученном выражении присутствует произведение $f * f_q$. В силу того, что f_q является обратным элементом для f по модулю q, произведение равно единице. В результате получим следующее выражение:

$a = p\phi * g + f * m \ (mod\ q)$

В результате применения операции центрирования коэффициентов и редукции по модулю p получим:

$m' = f * m \ (mod\ p)$

Для того, чтобы убрать многочлен f осталось домножить выражение на обратный элемент f_p.

$m'' = m \ (mod\ p)$

В результате получили исходное сообщение m, следовательно, алгоритм позволяет корректно дешифровать сообщение.

4.7 Атаки с использованием решетки

Предположим, что злоумышленнику известны параметры криптосистемы: N, p, q. А так же ему удалось перехватить публичный ключ h.

Рассмотрим квадратную матрицу размера $2N$, имеющую следующий вид:

$$\text{A} = \begin{pmatrix} \alpha & 0 & \dots & 0 & h_0 & h_1 & \dots & h_{N-1} \\ 0 & \alpha & \dots & 0 & h_{N-1} & h_0 & \dots & h_{N-2} \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \alpha & h_1 & h_2 & \dots & h_0 \\ 0 & 0 & \dots & 0 & q & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & q & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & 0 & 0 & 0 & \dots & q \end{pmatrix}$$

параметр α - небольшое целое число.

Пусть строки данной матрицы являются базисом целочисленной решетки L.

В первом столбце и в последней строке матрицы находятся $N-1$ нулей. Последовательно применяя теорему Лапласа для разложения определителя, можно вычислить детерминант данной матрицы: $det(A) = \alpha^N * q^N - 0 = \alpha^N q^N$

Из вида публичного ключа $h = g * f^{-1}$ следует, что решетка L содержит вектор $\tau = (\alpha f_1, \alpha f_2, ..., \alpha f_N, g_1, g_2, ..., g_N) = (\alpha f, g)$. Данный вектор является кратчайшим вектором решетки L.

Таким образом, злоумышленник теоретически может получить секретные ключи f и g, с помощью которых можно расшифровать неограниченное число сообщений при условии того, что ключи не обновляются. Возникает задача поиска кратчайшего вектора в решетке L.

Согласно эвристике Гаусса длина кратчайшего вектора может быть ограничена:
$det(A)^{\frac{1}{2N}} \sqrt{\frac{2N}{2\pi e}} < |v| < det(A)^{\frac{1}{2N}} \sqrt{\frac{2N}{\pi e}}; \quad \sqrt{\frac{N\alpha q}{\pi e}} < |v| < \sqrt{\frac{2N\alpha q}{\pi e}}$

Для того, чтобы вероятность получения τ или близкому к нему вектора с помощью алгоритма LLL была максимальной необходимо, чтобы отношение $k = \frac{\sqrt{\frac{N\alpha q}{\pi e}}}{|\tau|}$ было максимальным.

Сложность поиска кратчайшего вектора в таком случае растет экспоненциально от N. Несмотря на это, на практике осуществима успешная атака с использованием алгоритма LLL при условии небольшого N.

При использовании стандартных параметров, предлагаемых авторами данной криптосистемы атаки с помощью решеток являются трудноосуществимыми. Это следует из того, что алгоритм LLL в общем случае может дать лишь приближенное решение задачи о поиске вектора $(\alpha f, g)$, а большая вычислительная сложность не позволит совершать атаки на системы с периодически обновляемыми ключами.

Таблица 1 - Рекомендуемые параметры криптосистемы NTRU.

Уровень безопасности	N	p	q
Минимальный	167	3	128
Стандартный	251	3	128
Высокий	347	3	128
Высочайший	503	3	256

С помощью решетки можно также совершить атаку для получения вектора $(\alpha m, \phi)$, из которого можно получить исходное сообщение. Использование данного подхода неэффективно, т.к. может быть взломано лишь одно сообщение за большой промежуток времени.

4.8 Атаки перебором

Существует три способа перебора для взлома алгоритма:

1. Перебор $f \in M_f$

   Перебирая ключи f необходимо следить за коэффициентами $g = f * h \ (mod\ q)$, которые должны быть достаточно малыми.

2. Перебор $g \in M_g$

   При переборе ключей g получаем аналогичную ситуацию для ключа $f = g * h^{-1} \ (mod\ q)$, коэффициенты которого должны получаться небольшими.

3. Перебор $\phi \in M_\phi$

   В таком случае небольшими должны быть коэффициенты многочлена $e - \phi * h \ (mod\ q)$

Для взлома большого числа сообщений оптимальным выбором является перебор $g \in M_g$, т.к. число комбинаций меньше, чем во множестве M_f. Для взлома отдельного сообщения оптимальным будет перебор $\phi \in M_\phi$.

Несмотря на то, что атака полным перебором осуществима в теории, на практике данный способ является крайне неэффективным. Существует улучшенная версия атака перебором, которая называется встречей посередине. Для этого искомый ключ f разделяется на две части, например, пополам. Использование такого подхода дает некоторое ускорение за счет использования большого количества памяти.

4.9 Атака с замаскированным перехватом

Рассмотрим атаку, совершаемую во время обмена публичными ключами. Обмен публичными ключами производится по незащищенному каналу связи, поэтому злоумышленник может легко их перехватить.

Собеседник А отправляет публичный ключ собеседнику В, но ключ перехватывается злоумышленником С. Злоумышленник генерирует свои ключи - секретный и публичный. Публичный ключ злоумышленника отправляется собеседнику В. То же самое происходит и при отправке публичного ключа от В к А.

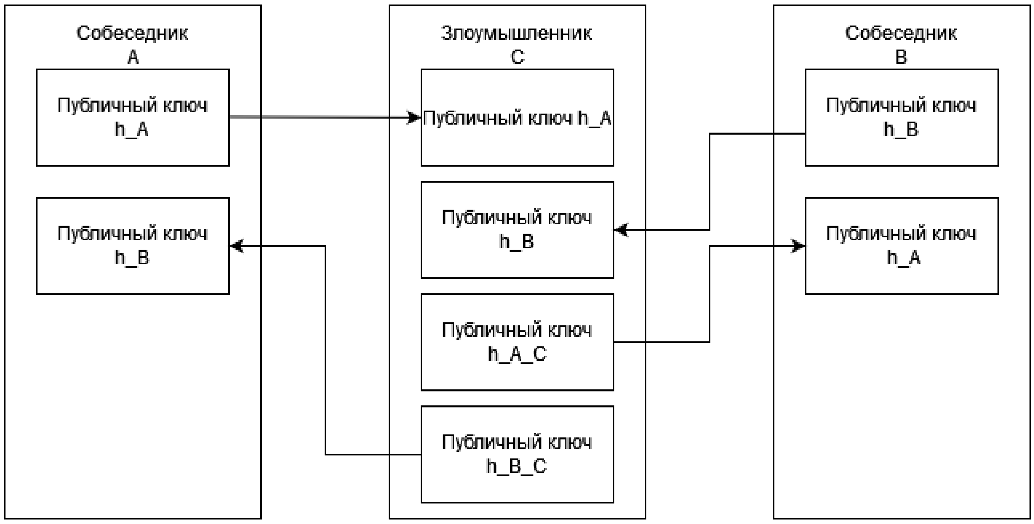

Рисунок 2: Схема перехвата публичных ключей.

Таким образом, злоумышленник становится замаскированным посредником между собеседниками А и В. Зашифрованное сообщение от А дешифруется злоумышленником С с помощью сгенерированного секретного ключа, затем шифруется с помощью публичного ключа, полученного от собеседника В. Таким же образом перехватываются и дешифруются сообщения от В к А.

В результате такой атаки обмен сообщениями между собеседниками не нарушается, они даже не подозревают о существовании перехватчика сообщений. Для защиты от подобных атак необходимо использовать процедуру проверки подлинности ключей.

5 Программная реализация схемы NTRUEncrypt

5.1 Постановка задачи и требования к программной системе

Требуется разработать программную систему, реализующую вышеописанную схему шифрования NTRU. Шифрование данных - задача, которая встречается в большом количестве программных продуктов. Исходя из этого, логично, что помимо готового приложения для пользователей, необходимо подумать и об интеграции с другими продуктами. По этой причине ключевые функции системы необходимо реализовать в отдельном модуле, с программным интерфейсом, который обеспечит такие интеграции.

В результате размышлений был выдвинут ряд требований к программной системе. Система должна:

1. Обеспечивать шифрование данных

Входные данные должны быть надежно зашифрованы с помощью алгоритма криптосистемы NTRU. Зашифрованные данные должны быть однозначно дешифрованы без потерь информации. Кроме этого необходимо обеспечить возможность генерации ключей и их загрузку.

2. Поддерживать шифрование текстовых файлов

Поскольку любой файл можно интерпретировать как текст, то необходимо реализовать поддержку текстовых файлов. Для этого необходимо разработать функционал, обеспечивающий чтение из файла и запись в файл сообщений и ключей. Для корректного шифрования больших файлов необходимо реализовать алгоритм разбиения текста на блоки, а также алгоритм соединения этих блоков.

3. Обеспечивать высокое быстродействие

Задача шифрования данных встречается повсеместно. Современные программы совершают огромное число соединений за единицу времени, поэтому суммарные накладные расходы на шифрование могут оказать существенное влияние на производительность. По этой причине криптосистема должна работать на столько быстро, на сколько это возможно.

4. Поддерживать режим работы в режиме CLI

Необходимо реализовать интерфейс командной строки для интеграции с командным интерпретатором операционной системы. Это необходимо для поддержки автоматизации действий. Система должна быть способна взаимодействовать вместе с основными скриптовыми языками, являющимися стандартом для автоматизации в современных ОС. Примерами таких языков являются bash, batch, powershell, и другие.

5. Иметь графический пользовательский интерфейс

Для рядовых пользователей, желающих обеспечить конфиденциальность своих данных система должна иметь привычный им дружелюбный графический пользовательский интерфейс.

5.2 Используемые технологии

5.2.1 Язык программирования

Одно из главных требование к программе - высокая производительность, поскольку современные приложения могут отправлять огромное число сетевых запросов за кратчайший промежуток времени, в таком случае время работы алгоритмов криптографии сильно сказывается на производительности всей системы в целом.

В качестве основного языка программирования был выбран C++. Благодаря компиляции в настоящий машинный код, статической типизации и большому числу оптимизаций компилятора, C++ является оптимальным выбором для решения данной задачи.

Кроме того, использования данного языка с описанными далее библиотеками обеспечивает практически полную кроссплатформенность решения - способность работать на большинстве программных и аппаратных платформах.

5.2.2 Библиотека для работы с многочленами

Для работы с многочленами, кольцами и т.д. была выбрана библиотека NTL. Выбор библиотеки обусловлен высокой производительностью, а так же тем, что библиотека содержит все необходимые для реализации алгоритмы работы с многочленами и операциями по модулю. Кроме того, библиотека является свободной с точки зрения лицензии. Библиотека распространяется под лицензией открытого исходного кода LGPLv2.1+.

Стоит отметить, что для сборки библиотеки с помощью компилятора Microsoft Visual C++ потребовались сделать небольшие модификации ее кода:

1. Включить поддержку C++ исключений, т.к. по умолчанию она выключена. Это необходимо для обеспечения стабильности работы программы, т.к. при использовании подхода с кодами ошибок вероятность пропустить некорректное поведение программы гораздо выше. Для этого в заголовочном файле config.h потребовалось изменить директиву препроцессора.

```
#if 1
#define NTL_EXCEPTIONS

```

2. Выключить предупреждение компилятора 4146 в заголовочном файле lip.h.

```
#pragma warning ( disable : 4146 )

```

5.2.3 Фреймворк для разработки пользовательского интерфейса

Для разработки графического пользовательского интерфейса был выбран фреймворк Qt. Qt - кроссплатформенный инструмент, позволяющий быстро и удобно разработать интерфейс с помощью графического редактора форм. Данный инструмент предоставляет возможность расположить графические компоненты на форме и автоматически генерировать прототипы обработчиков событий, внутри которых остается только описать действия, которые должны выполняться при получении сигналов.

7.4 Схема программной системы

Вначале была построена схема компонентов программной системы, на которую можно будет ориентироваться при разработке. Данная схема отражает взаимосвязь компонентов, а также их основную функциональность.

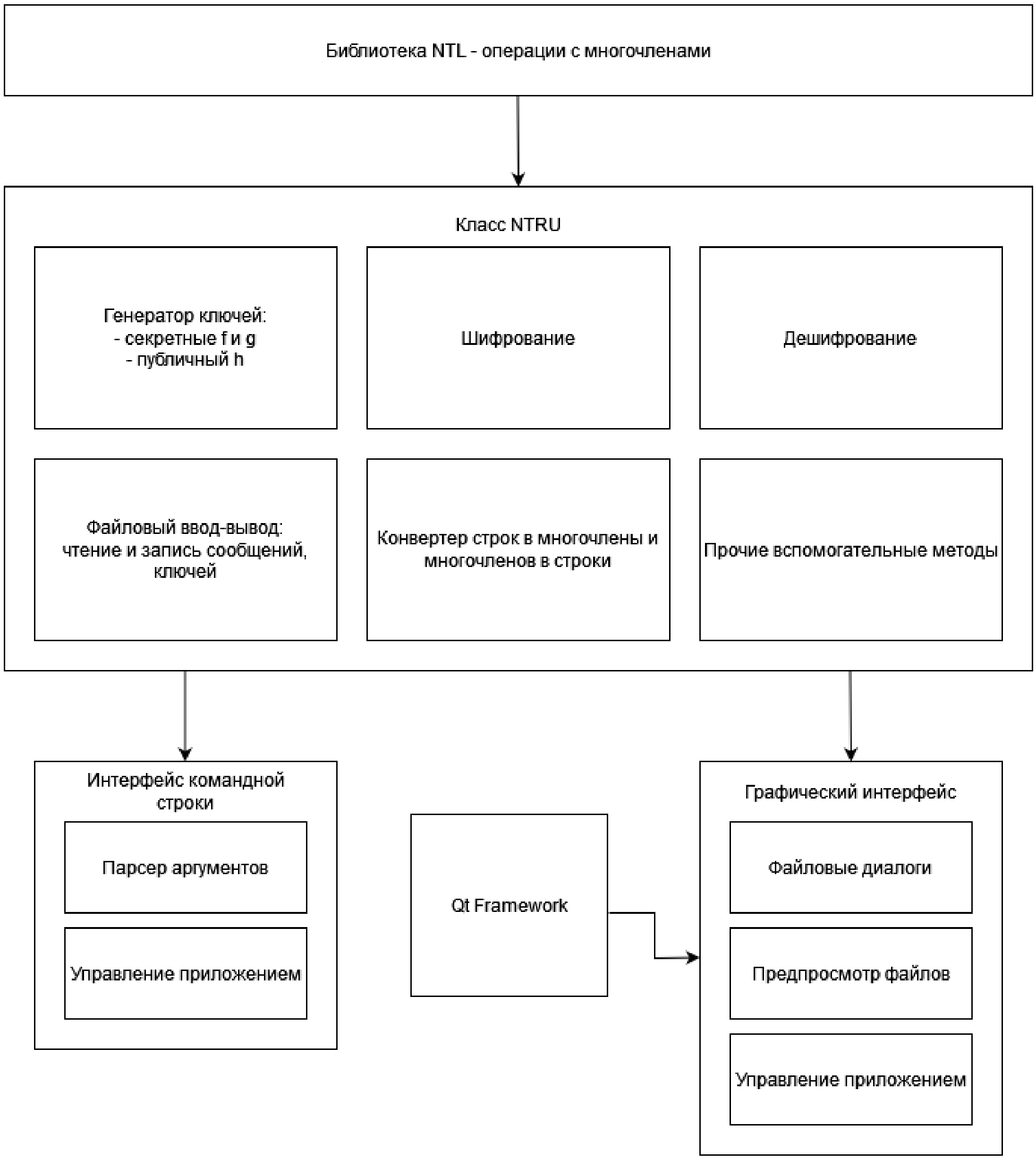

Рисунок 3: Схема компонентов системы.

5.3 Реализация класса, обеспечивающего шифрование

5.3.1 Генерация ключей

Для генерации секретных ключей f и g создается вектор случайных коэффициентов из множества {-1, 0, 1}. Значения генерируются исходя из соотношения, описанного в пункте 4.3. Затем вектор преобразуется в многочлен NTL::ZZX.

Полученные многочлены f и g проверяются на возможность инвертирования. Если обратный многочлен невозможно найти, то генерируется новая пара.

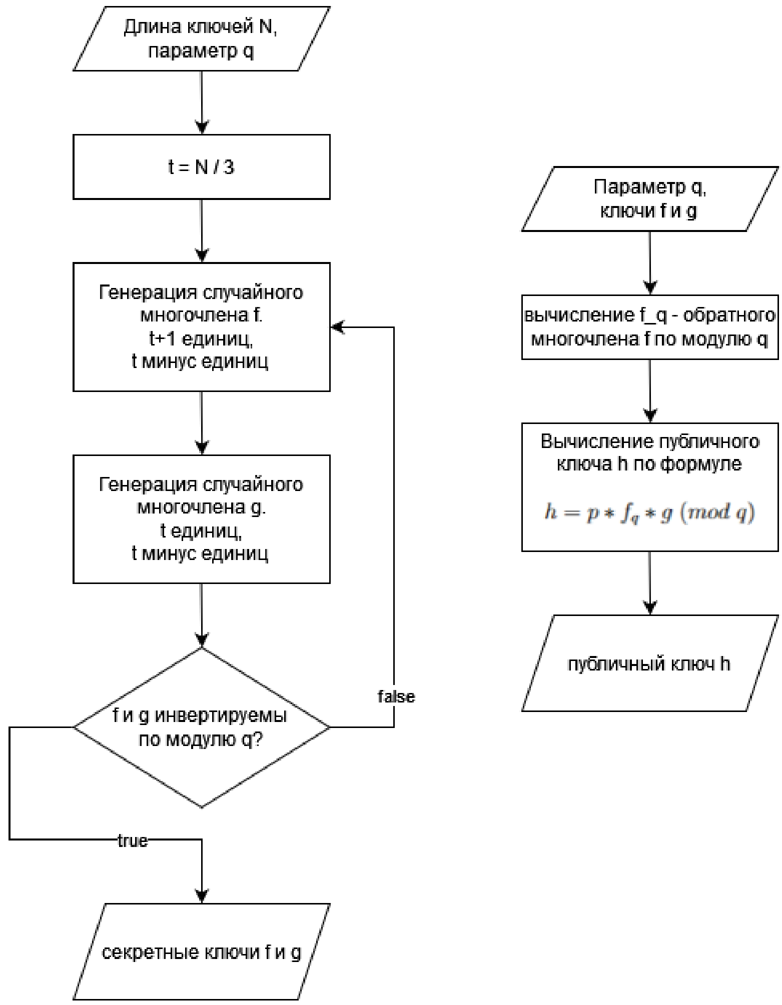

Рисунок 4: Схема алгоритмов генерации ключей.

Публичный ключ вычисляется по формуле $h = p * f_q * g \ (mod\ q)$.

5.3.2 Шифрование

Примечание: рассматривается реализация алгоритма шифрования небольших строк. В случае длинной строки предварительно производится разбиение на блоки.

За шифрование исходного сообщения отвечает метод класса
NTL::ZZ_pX NTRU::encrypt_str(std::string str);

Во-первых, необходимо преобразовать строку в код с алфавитом {-1, 0, 1}. Данная строка представляется как вектор кодов символов таблицы ASCII, т.е. важна именно числовая интерпретация. Далее каждый элемент вектора кодов ASCII преобразуется в троичную систему исчисления, так получается небольшой вектор для каждого символа. Затем все эти векторы соединяются в единый вектор, элементы которого удовлетворяют условию (троичный код): $\forall v[i] \in v : \ v[i] \in \{0,1,2\}$. Затем код "сдвигается"на единицу влево, т.е. в результате получается сообщение в алфавите {-1, 0, 1}.

После конвертации строки в код из полученных коэффициентов создается многочлен типа NTL::ZZX.

Для шифрования требуется случайный многочлен r с коэффициентами из множества $\{-1,0,1\}$. Для генерации такого многочлена используется псевдослучайный генератор Твистера Мерсенна из 32-битных чисел с размером состояния 19937 бит - mt19937.
NTL::ZZ_pX r = cvt_ZZX_to_ZZ_pX(random_polynomial(N, N / 3, N / 3));

Перед совершением операций над многочленами необходимо настроить систему на работу по необходимому модулю, в данном случае это модуль q.
NTL::ZZ_p::init(NTL::ZZ(q));
Zx_Ring = NTL::ZZ_pX(NTL::INIT_MONO, N) - 1;

С помощью функции NTL::MulMod вычисляется произведение многочленов r и h по модулю q.
NTL::ZZ_pX rh = NTL::MulMod(r, h, Zx_Ring);

Сообщение конвертируется м многочлен типа NTL::ZZ_pX.
NTL::ZZ_pX m = cvt_ZZX_to_ZZ_pX(message);

Вычисляется многочлен e - зашифрованное сообщение.
NTL::ZZ_pX e = (p * rh) + m;

Таким образом, исходный текст шифруется в многочлен типа NTL::ZZ_pX, который далее может быть сохранен в файл или каким-то образом использован в коде.

5.3.3 Дешифрование

Примечание: рассматривается реализация алгоритма дешифрования небольших строк. В случае длинной строки предварительно производится разбиение на блоки.

Дешифрование происходит в методе класса

std::string NTRU::decrypt_str(NTL::ZZ_pX encrypted_str_poly)

Сообщение расшифровывается по формулам:

$a = f * e \ (mod\ q)$; $m = fp * a \ (mod\ p)$

Во первых, для дешифрования сообщения необходимо конвертировать многочлен f типа NTL::ZZX в многочлен по модулю q f_q типа NTL::ZZ_pX.

NTL::ZZ_pX f_q = cvt_ZZX_to_ZZ_pX(f);

Затем вычисляется многочлен $a = f * e$.

NTL::ZZ_pX a_tmp = NTL::MulMod(f_q, encrypted_message, Zx_Ring);

Производится конвертирование многочлена a из типа NTL::ZZ_pX в тип NTL::ZZX.

NTL::ZZX a = cvt_ZZ_pX_to_ZZX(a_tmp);

Для дальнейших действий класс настраивается на работу с многочленами по модулю p.

NTL::ZZ_p::init(NTL::ZZ(p));

Zx_Ring = NTL::ZZ_pX(NTL::INIT_MONO, N) - 1;

Производится конвертирование многочленов a и f типа NTL::ZZX в многочлены по модулю p a_p и f_p типа NTL::ZZ_pX.

NTL::ZZ_pX a_p = cvt_ZZX_to_ZZ_pX(a);

NTL::ZZ_pX f_p = cvt_ZZX_to_ZZ_pX(f);

На последнем шаге вычисляется многочлен m, который является расшифрованным сообщением.

NTL::ZZX m = cvt_ZZ_pX_to_ZZX(NTL::MulMod(InvMod(f_p, Zx_Ring), a_p, Zx_Ring));

Таким образом, зашифрованный текст дешифруется в многочлен типа NTL::ZZX, который далее может быть сохранен в файл или каким-то образом использован в коде (например, может быть соединен с остальными блоками при блочном делении строк и преобразован в текстовый вид).

5.3.4 Ускорение алгоритмов с помощью параллельных вычислений

Современные процессоры имеют как минимум два ядра. Следовательно, имеет смысл попытаться распараллелить часть основных алгоритмов программы. Было проведено распараллеливание алгоритмов шифрования и дешифрования.

Для ускорения алгоритма шифрования строка, содержащая исходный текст, разбивается на блоки. По умолчанию размер блока равен 12. Затем к полученным блокам применяется алгоритм шифрования. Данная операция выполняется параллельно в цикле благодаря директиве #pragma omp parallel for из библиотеки OpenMP. После этого зашифрованные блоки упорядоченно записываются в файл, который является результатом работы алгоритма.

Параллельный алгоритм дешифровния основан на том же принципе. Исходя из алгоритма шифрования, зашифрованное сообщение уже разбито на блоки. Блоки в параллельном цикле дешифруются и раскодированные части сообщения соединяются в результирующую строку, не нарушая порядка следования.

Такой подход распараллеливания алгоритмов позволил с минимальными модификациями исходного кода существенно повысить производительность программы. Сравнение последовательных и параллельных алгоритмов приведено в разделе «Тестирование».

5.4 Реализация пользовательского интерфейса

5.4.1 Интерфейс командной строки (CLI)

Для автоматизации действий с помощью средств операционной системы (командный интерпретатор) было принято решения добавить поддержку интерфейса командной строки.

Для работы с программой в режиме командной строки был реализован парсер аргументов, способный работать со следующими аргументами:

1. NTRU.exe keys <key f file> <key g file> <key h file>
2. NTRU.exe enc <source file> <key h file> <encrypted file>
3. NTRU.exe dec <encrypted file> <key f file> <key g file> <decrypted file>

Парсер аргументов проверяет их количество. Если количество меньше одного, то выводится сообщение об ошибке, содержащее справочную информацию о корректных аргументах.

Если в качестве первого аргумента передано слово <keys>, то происходит вызов методов генерации ключей. Полученные ключи записываются в файлы, пути к которым являются следующими тремя аргументами: <key f file>, <key g file>, <key h file>.

Если первым аргументом является слово <enc>, то запускается алгоритм шифрования, на вход которому поступает текст из файла по пути <source file>, шифрование производится с помощью публичного ключа <key h file>. Зашифрованный текст сохраняется в файл по пути <encrypted file>.

Если первым аргументом было передано слово <dec>, то зашифрованный текст из файла <encrypted file> передается на вход алгоритма дешифрования, который с помощью секретных ключей по путям <key f file> и <key g file> декодирует сообщение. Полученный текст сохраняется в файл по пути <decrypted file>.

Если первый аргумент не принадлежит множеству {keys, enc, dec} или в процессе разбора следующих алгоритмов возникнет исключение, то в консоль выводится сообщение об ошибке и список правильных аргументов командной строки.

5.4.2 Графический интерфейс

Графический интерфейс пользователя должен быть удобным и понятным. Для этого был использован фреймворк Qt.

Алгоритм работы приблизительно такой же как и консольной версии. Единственное отличие заключается в том, что вместо аргументов используются графические компоненты такие как файловые диалоги и кнопки, которые связаны с обработчиками сигналов - методами класса интерфейса, в которых вызываются методы класса NTRU.

Интерфейс был разработан с помощью графического редактора форм Qt designer на основе следующего макета.

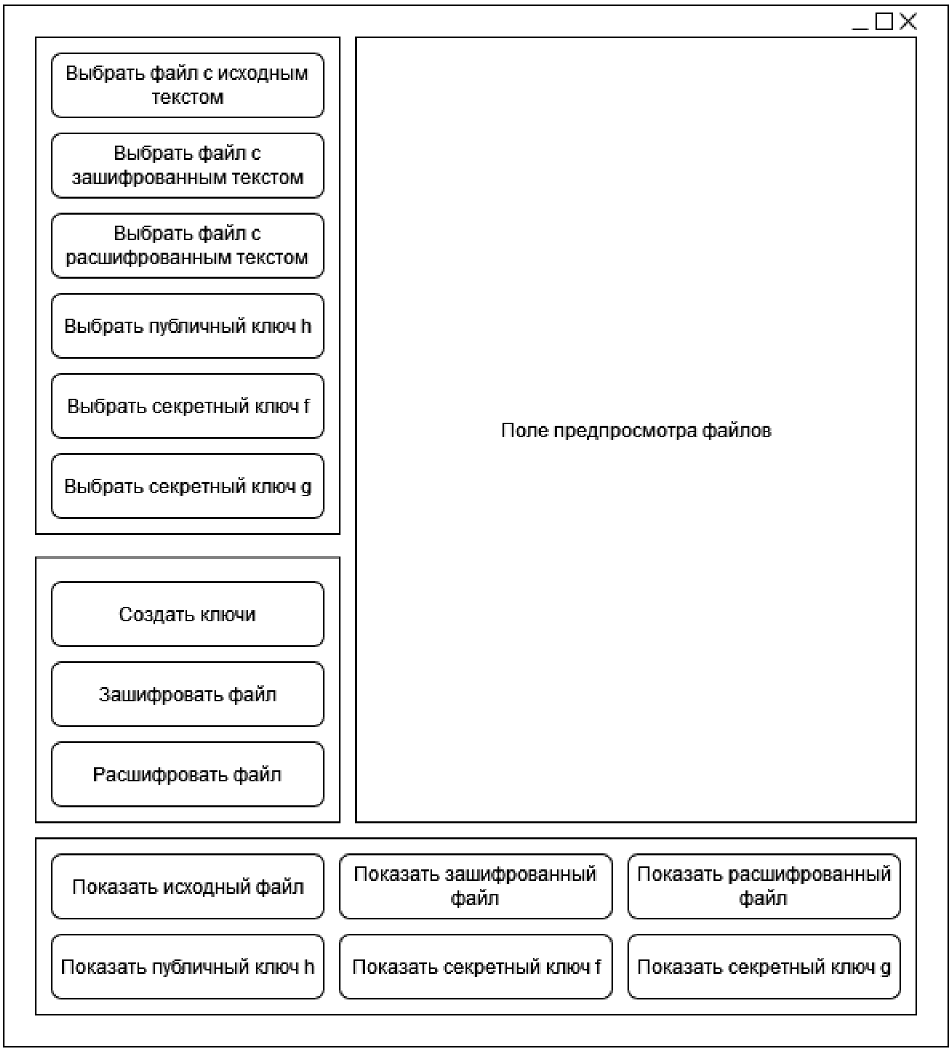

Рисунок 5: Приблизительная схема графического интерфейса.

5.5 Тестирование программы

Разработанный класс NTRU был протестирован при различных параметрах криптосистемы. На каждом наборе параметров было проведено тестирование с помощью случайных строк различной длины. Так же были протестированы методы для работы с файлами. Стоит отметить, что разработка велась по технологии TDD - написание тестов до реализации алгоритмов, для автоматизации тестирования был использован инструмент, доступный на GitHub - yaml скрипты.

Кроме тестирования корректности работы алгоритмов программы необходимо также провести тесты производительности. Замеры времени были проведены с использованием функций из стандартной библиотеки - std::chrono. Тестирование проводилось на компьютере, обладающем следующими характеристиками:

1. Центральный процессор: Intel i5-10400F, 6 ядер, 12 потоков, 4.01 GHz
2. Оперативная память: Kingston HyperX DDR4, 2666 MHz
3. SSD накопитель: Samsung 980 NVMe, чтение 3500 Mb/s, запись 3000 Mb/s

Для оценки производительности были проведены замеры времени основных алгоритмов программы. Для автоматизации тестирования были использованы скрипты на языке Python, с помощью которых результаты были записаны в удобном для последующего анализа виде. Было проведено тестирование времени работы в зависимости от параметра N (тестировались рекомендуемые параметры) и от количества символов в исходном файле. Ниже приведены таблицы, содержащие результаты бенчмарков (для последовательных и параллельных алгоритмов), а также приведены графики, отражающие результаты. Время измеряется в миллисекундах.

Таблица 2 - Зависимость времени работы алгоритмов от размера ключа.

N	Ключи	Шифрование посл.	Шифрование пар.	Дешифрование посл.	Дешифрование пар.
167	3	783	126	651	108
251	4	1147	225	972	175
347	6	2325	392	1537	255
503	8	5749	943	2153	358

Таблица 3 - Зависимость времени работы алгоритмов от числа символов в файле.

Число символов	Шифрование посл.	Шифрование пар.	Дешифрование посл.	Дешифрование пар.
50	21	7	27	7
100	38	8	47	7
500	178	34	175	27
1000	359	73	328	52
2000	743	134	623	109
3000	1111	203	923	161
4000	1458	258	1245	240
5000	1779	315	1510	283

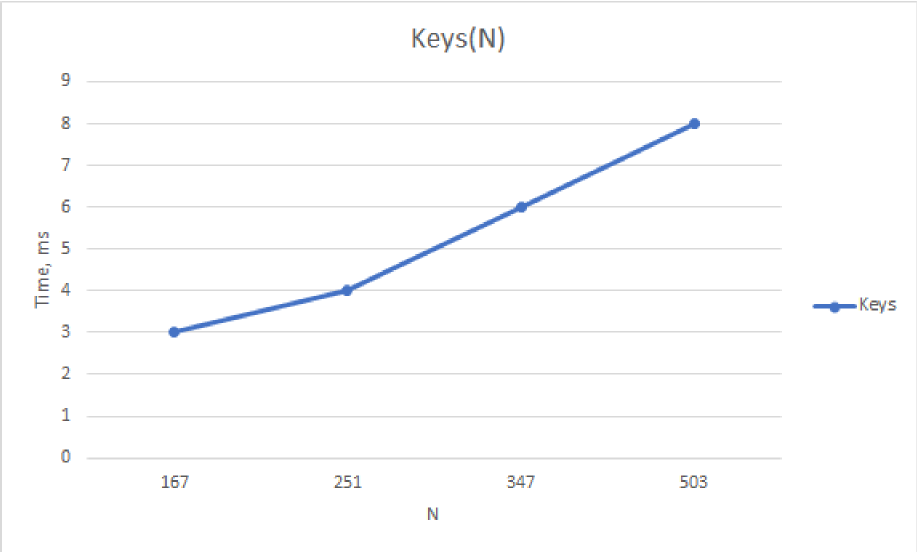

Рисунок 6: Зависимость времени генерации ключей от параметра N

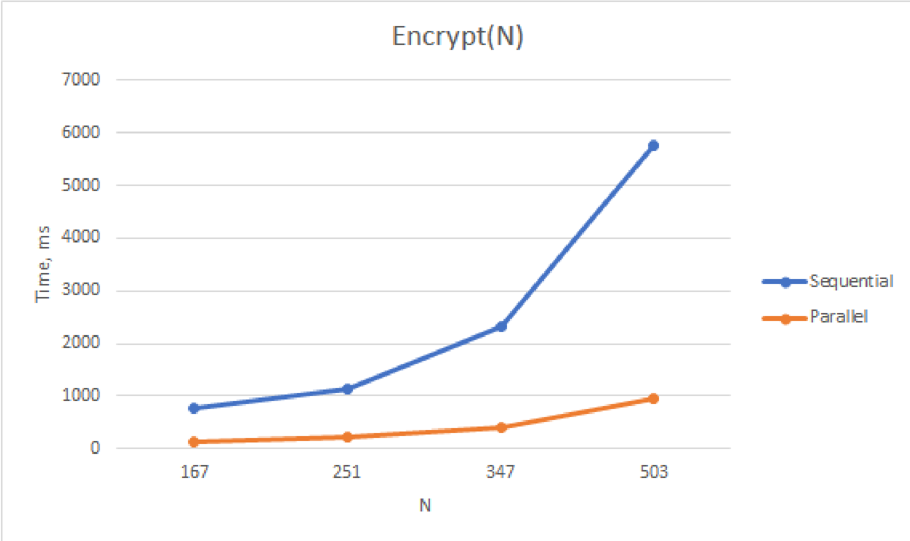

Рисунок 7: Зависимость времени шифрования от параметра N

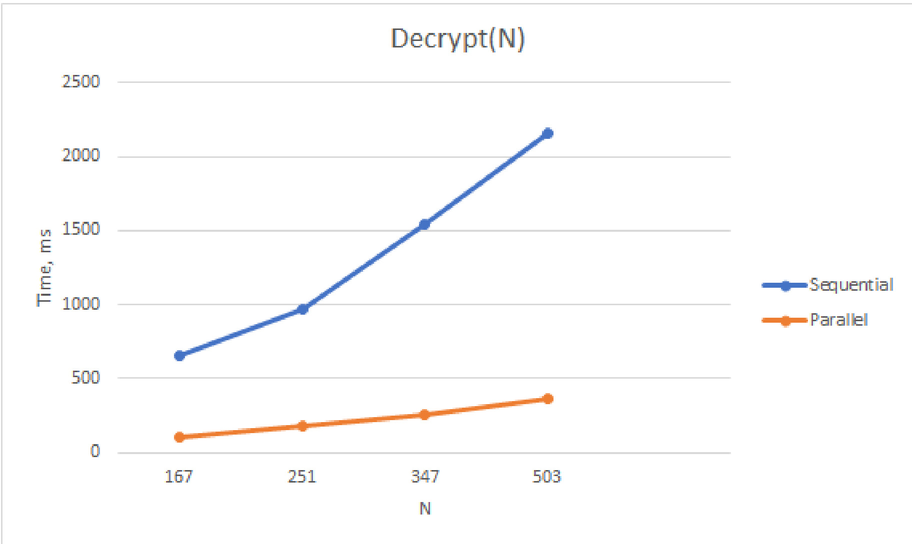

Рисунок 8: Зависимость времени дешифрования от параметра N

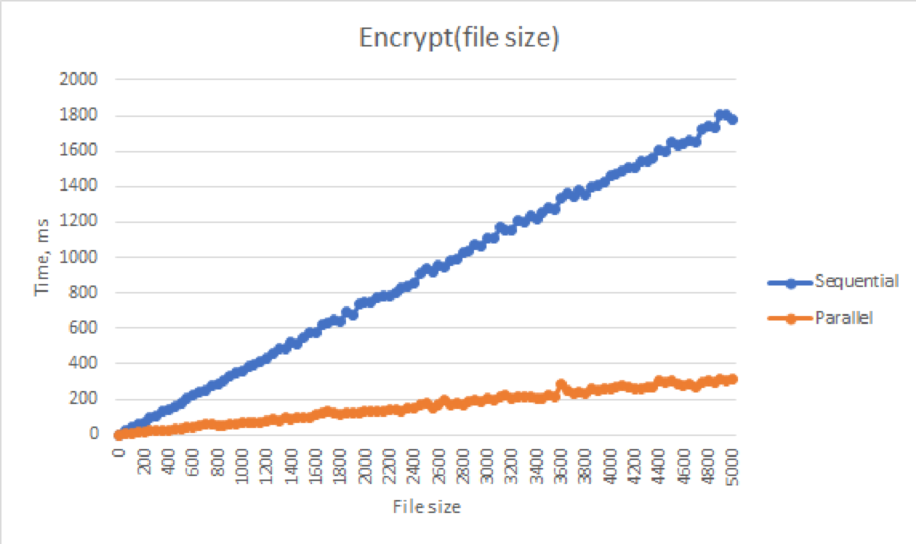

Рисунок 9: Зависимость времени шифрования от числа символов в файле

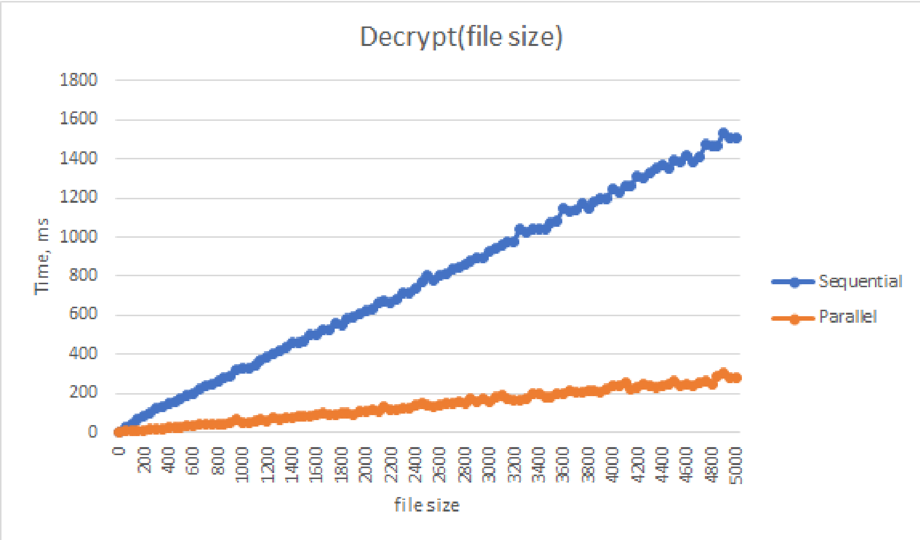

Рисунок 10: Зависимость времени дешифрования от числа символов в файле

Исходя из результатов, можно сделать вывод, что зависимость от параметра N является полиномиальной. Зависимость времени от размера исходного файла оказалась линейной. Кроме того, стоит отметить, что распараллеливание с помощью библиотеки OpenMP привело к ощутимому приросту производительности программы. Время работы по сравнению с последовательной версией уменьшилось в 5-6 раз. Как было сказано ранее, тестирование проводилось на 12-ти потоках. Этот параметр можно легко изменить не меняя исходный код самой программы - необходимо создать переменную окружения OMP_NUM_THREADS. В частности, этот способ может помочь оптимизировать работу приложения на серверах.

5.6 Руководство пользователя

5.6.1 Интерфейс командной строки

При запуске бинарного файла программы без аргументов или с ошибкой в аргументов выводится сообщение с описанием всех доступных аргументов командной строки.

Для генерации ключей необходимо ввести аргумент keys, а так же пути сохранения файлов с созданными ключами f (секретный), g (секретный) и h (публичный).

Для шифрования файла необходимо ввести аргумент enc, путь к файлу, путь к публичному ключу h, путь для сохранения зашифрованного файла.

Для дешифрования файла необходимо ввести аргумент dec, путь к зашифрованному файлу, путь к секретному ключу f, путь к секретному ключу g, путь для сохранения расшифрованного файла.

Ниже приведен пример использования всех возможностей программы в режиме командной строки. Строка 1 - помощь, строка 7 - создание ключей, строка 11 - шифрование, строка 13 - дешифрование.

```
1 C:\Users\veter\source\NTRU_UNN\NTRU\Release>NTRU.exe
2 Args error. Expected:
3         keys <f_file> <g_file> <h_file>
4         enc <input_file> <h_file> <output_file>
5         dec <input_file> <f_file> <g_file> <output_file>
6
7 C:\Users\veter\source\NTRU_UNN\NTRU\Release>NTRU.exe keys f.txt g.txt
     h.txt
8
9 C:\Users\veter\source\NTRU_UNN\NTRU\Release>type input.txt
10 Hello, world!
11 C:\Users\veter\source\NTRU_UNN\NTRU\Release>NTRU.exe enc input.txt h.
    txt enc.txt
12
13 C:\Users\veter\source\NTRU_UNN\NTRU\Release>NTRU.exe dec enc.txt f.
    txt g.txt dec.txt
14
15 C:\Users\veter\source\NTRU_UNN\NTRU\Release>type dec.txt
16 Hello, world!
17
```

5.6.2 Графический интерфейс

Для генерации ключей необходимо с помощью кнопок, вызывающих файловые диалоги выбрать файлы для сохранения ключей f, g, h. После выбора файлов необходимо нажать кнопку Create keys.

Для шифрования файла необходимо с помощью кнопок, вызывающих файловые диалоги выбрать файлы для загрузки исходного файла, ключа h, а также для сохранения зашифрованного файла. После выбора файлов необходимо нажать кнопку Encrypt.

Для дешифрования файла необходимо с помощью кнопок, вызывающих файловые диалоги выбрать файлы для загрузки зашифрованного файла, ключа f, ключа g, а также для сохранения дешифрованного файла. После выбора файлов необходимо нажать кнопку Decrypt.

Так же в графическом интерфейсе реализована возможность предпросмотра файлов. Для этого в разделе Show необходимо выбрать файл для отображения.

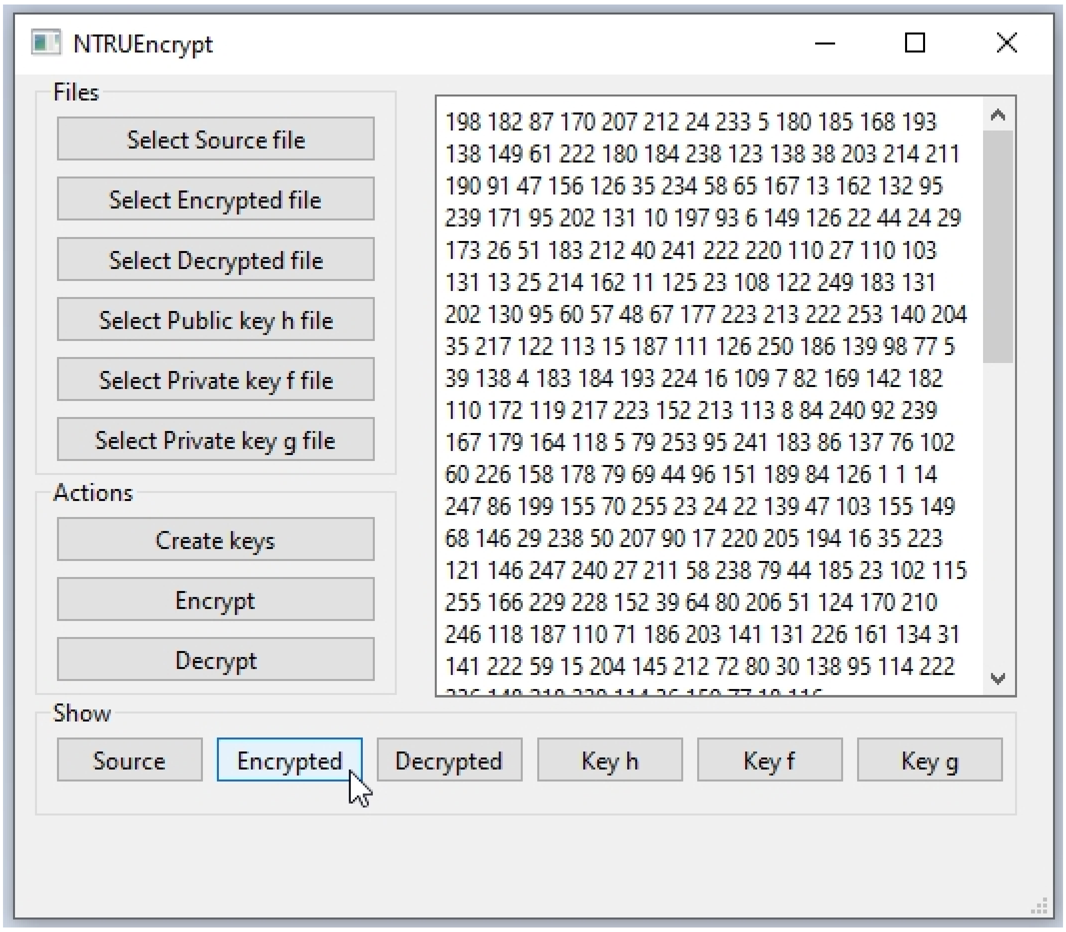

Рисунок 11: Внешний вид графического интерфейса.

Заключение

В результате изучения данной темы была достигнута основная цель работы - изучение алгоритмов шифрования с использованием решеток а также сложных математических задач, которые лежат в их основе.

Во-первых, были изучены основные понятия криптографии и системы шифрования с открытым ключом. Было рассмотрено понятие односторонней функции, которое является основой современной криптографии. Так же я выяснил, что в настоящее время шифрование применяется практически во всех сферах деятельности. Особенно актуально это стало с активным внедрением цифровой экономики.

Была рассмотрена задача о ближайшем векторе решетки. Смысл данной задачи заключается в нахождение вектора решетки, который находится как можно ближе к заданному. Я рассмотрел итерационный метод решения данной задачи - алгоритм ближайшей плоскости Бабая. Выяснил, что основным недостатком алгоритма является использование ортогонализации Грама-Шмидта. Но, как оказалось, процесс можно ускорить с помощью использования LLL редуцированного базиса. Кроме данного алгоритма рассмотрел метод округления, который так же позволяет найти приближенное решение задачи ближайшего вектора. Преимуществом этого алгоритма является высокая скорость работы, которая зависит только от алгоритма решения СЛАУ. Затем изучил то, как можно свести задачу о ближайшем векторе решетки к задаче о кратчайшем векторе решетки. Этот алгоритм называется методом вложения. Идея заключается в отыскании кратчайшего вектора, который является разницей между заданным вектором и ближайшим вектором решетки.

Была изучена задача о кратчайшем векторе решетки. Кратчайший вектор решетки - вектор наименьшей длины, не равный нулю. В ходе исследования этой проблемы было выяснено, что ортогональном базисе это решается элементарно. Чтобы свести задачу к более простой были изучены алгоритмы, которые приводят базис решетки в вид, который близок к ортогональному. Эти алгоритмы имеют название Лагранжа-Гаусса и Ленстры-Ленстры-Ловаса (LLL). Наименьший вектор полученного базиса может служить приближенным решением задачи о кратчайшем векторе. В ходе исследования проблемы были даны оценки решения, на сколько оно отличается от точного. Кроме того, было установлено, что редуцированный базис может быть полезен в других алгоритмах, выступая в качестве начального.

После этого было проведено изучение системы шифрования GGH. Было выяснено, что алгоритм основан на поиске ближайшего вектора решетки в заданном базисе. Идея шифрования заключается в том, что сообщение шифруется с помощью ”плохого” базиса решетки, который состоит из длинных и далеких от ортогональности векторов. При использовании большой размерности решетки данный ключ может быть безопасно передан по незащищенному каналу связи. Идея дешифрации заключается в поиске ближайшего вектора в ”хорошем” базисе, состоящем из коротких и близких к ортогональности векторов. Использование такого базиса позволяет эффективно расшифровать сообщение.

Затем в ходе работы была изучена система шифрования NTRUEncrypt. Алгоритм основан на кольцах усеченных многочленов. Криптографическая стойкость данной системы обеспечивается трудной вычислительной задачей о поиске кратчайшего вектора решетки. Анализ возможных атак показал, что алгоритм является надежным при условии использования достаточно большой степени многочленов и использовании рекомендуемых параметров операций по модулям.

После изучения теоретической части алгоритмов была разработана программа, демонстрирующая работу алгоритмов системы NTRUEncrypt. Алгоритм был реализован на языке программирования C++. Для интеграции со скриптовыми языками был реализован интерфейс командной строки. Также было реализован графический пользовательский интерфейс, позволяющий привычном для большинства пользователей способом взаимодействовать с программой. Программа была протестирована при различных конфигурациях и различных входных данных. Благодаря использованию параллельных вычислений удалось ускорить работу некоторых алгоритмов, что привело к существенному повышению быстродействия. Ускорение по сравнению с последовательной реализацией составило 5-6 раз при использовании 12 потоков на процессоре с 6-ю физическими ядрами. Хорошая производительность разработанной библиотеки шифрования приводит к заключению о том, что продукт может быть использован для обеспечения безопасности настоящих программных продуктов.

Список использованных источников и литературы

1. Шокуров А.В, Кузюрин Н.Н, Фомин С.А, Решетки, алгоритмы и современная криптография

2. Jeffrey Hoffstein, Jill Pipher, Joseph H. Silverman, NTRU: A Ring-Based Public Key Cryptosystem

3. Seong-Hun Paeng, Bae Eun Jung, and Kil-Chan Ha, A Lattice Based Public Key Cryptosystem Using Polynomial Representations

4. Joseph H. Silverman, NTRU and Lattice-Based Crypto: Past, Present, and Future

5. S. D. Galbraith, Mathematics of public key cryptography, Cambridge University Press, April 2012

6. Abderrahmane Nitaj, The Mathematics of the NTRU Public Key Cryptosystem

7. NTL: A Library for doing Number Theory [Электронный ресурс] - Режим доступа: https://libntl.org/doc/tour.html

8. Xagawa D. K. Cryptography with lattices. – 2010

9. De Micheli G., Heninger N., Shani B. Characterizing overstretched NTRU attacks //Journal of Mathematical Cryptology. – 2020. – Т. 14. – №. 1. – С. 110-119

10. Комарова А. В. и др. Теоретические возможности комбинирования различных математических примитивов в схеме электронной цифровой подписи //Кибернетика и программирование. – 2017. – №. 3. – С. 80-92

Приложения

Приложение 1. Исходный код класса шифрования NTRU.h

```
#pragma once
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <NTL/ZZ_pE.h>
#include <NTL/ZZX.h>
#include <NTL/GF2X.h>
#include <sstream>
#include <fstream>
#include <iomanip>
#include <cassert>
#include <numeric>

// helper function that prints a vector
template <typename T>
void print_vec(std::vector<T> vec) {
  for (int i = 0; i < vec.size(); i++) std::cout << vec[i] << " ";
  std::cout << std::endl;
}

// split string to blocks
std::vector<std::string> split_string_to_blocks(std::string str, int
    block_size);

class NTRU {

public:
  // Counstructor. N - deg, _p, _q - params
  NTRU(int _N, int _p, int _q);

  // encrypt string message to poly
  NTL::ZZ_pX encrypt_str(std::string str, bool check = true);

  // decrypt poly message to string
  std::string decrypt_str(NTL::ZZ_pX encrypted_str_poly);

  // blocked (and parallel) encrypt string message to poly
  std::vector<NTL::ZZ_pX> blocked_encrypt_str(std::string str, bool
    check = true, int block_size = 12);

  // blocked (and parallel) decrypt poly message to string
```

```
std::string blocked_decrypt_str(std::vector<NTL::ZZ_pX> encrypted_blocks);

// blocked (seq) encrypt string message to poly
std::vector<NTL::ZZ_pX> blocked_encrypt_str_seq(std::string str, bool check = true, int block_size = 12);

// blocked (seq) decrypt (seq) poly message to string
std::string blocked_decrypt_str_seq(std::vector<NTL::ZZ_pX> encrypted_blocks);

// save private key f to file private_f.txt
void save_private_f_to_file(const char* filename = "private_f.txt");

// save private key g to file private_g.txt
void save_private_g_to_file(const char* filename = "private_g.txt");

// save public key h to file public_h.txt
void save_public_h_to_file(const char* filename = "public_h.txt");

// save encrypted message to file encrypted.txt
void save_encrypted_to_file(NTL::ZZ_pX encrypted, const char* filename = "encrypted.txt");

// save encrypted blocked message to file encrypted.txt
void blocked_save_encrypted_to_file(std::vector<NTL::ZZ_pX> encrypted_blocks, const char* filename = "encrypted.txt");

// read private key f from file private_f.txt
void load_private_f_from_file(const char* filename = "private_f.txt");

// read private key g from file private_g.txt
void load_private_g_from_file(const char* filename = "private_g.txt");

// read public key h from file public_h.txt
void load_public_h_from_file(const char* filename = "public_h.txt");

// read encrypted message from file encrypted.txt
NTL::ZZ_pX load_encrypted_from_file(const char* filename = "encrypted.txt");

// read encrypted blocked message from file encrypted.txt
std::vector<NTL::ZZ_pX> blocked_load_encrypted_from_file(const char* filename = "encrypted.txt");

// create private keys f and g
```

```
  void create_private_keys();

  // calculate public key h
  void create_public_key();

private:
  int N, p, q; // NTRU params
  NTL::ZZX f; // private key
  NTL::ZZX g; // private key
  NTL::ZZ_pX h; // public key
  NTL::ZZ_pX Zx_Ring; // ring

  // convert char to ternary code vector
  std::vector<int> cvt_char_to_3_code(char c);

  // convert string to polynomial coefficients vector (-1, 0, 1)
  std::vector<int> cvt_string_to_polynomial_coeffs(std::string str);

  // convert ternary code vector to char
  char cvt_3_code_to_char(std::vector<int> char_3_code);

  // convert polynomial coefficients vector to string
  std::string cvt_polynomial_coeffs_to_string(std::vector<int> poly_coeffs);

  // get random polynomial coeffs vector (-1, 0, 1)
  std::vector<int> random_polynomial_coeffs(int n);

  // convert coeffs vector to NTL polynomial
  NTL::ZZX cvt_coeffs_vec_to_ntl_polynomial(std::vector<int> coeffs);

  // convert NTL polynomial to coeffs vector
  std::vector<int> cvt_ntl_polynomial_to_coeffs_vec(NTL::ZZX poly);

  // random poly deg N with count_positive 1, count_negative -1
  NTL::ZZX random_polynomial(int N, int count_positive, int count_negative);

  // test poly can be invert
  void self_test_invert();

  // test correct
  void self_test_equals();

  // covert ZZX to ZZ_pX
  NTL::ZZ_pX cvt_ZZX_to_ZZ_pX(NTL::ZZX poly);

  // covert ZZ_pX to GF2X
  NTL::GF2X cvt_ZZ_pX_to_GF2X(NTL::ZZ_pX poly);

  // convert GF2X to ZZ_pX
```

```
    NTL::ZZ_pX cvt_GF2X_to_ZZ_pX(NTL::GF2X poly);

    // convert ZZ_pX to ZZX
    NTL::ZZX cvt_ZZ_pX_to_ZZX(NTL::ZZ_pX poly);

    // invert poly, before calling, select a module for the ring
    NTL::ZZ_pX invert_poly(NTL::ZZ_pX poly);

    // encrypt NTL::ZZX message
    NTL::ZZ_pX encrypt(NTL::ZZX message, bool check);

    // decrypt NTL::ZZ_pX encrypted message
    NTL::ZZX decrypt(NTL::ZZ_pX encrypted_message);
};

```

Приложение 2. Исходный код класса шифрования NTRU.cpp

```
#include "NTRU.h"
#include <random>

std::vector<int> slice(const std::vector<int>& v, int start = 0, int
    end = -1) {
    int oldlen = v.size();
    int newlen;
    if (end == -1 || end >= oldlen) { newlen = oldlen - start; }
    else { newlen = end - start; }
    std::vector<int> nv(newlen);
    for (int i = 0; i < newlen; i++) {
        nv[i] = v[start + i];
    }
    return nv;
}

std::vector<std::string> split_string_to_blocks(std::string str, int
    block_size) {
    assert(str.length() >= block_size);
    std::vector<std::string> blocks;
    int block_count = str.length() / block_size + (str.length() %
    block_size == 0 ? 0 : 1);
    for (int i = 0; i < block_count; i++) {
        if (i * block_size > str.length()) {
            block_size = str.length() - (i - 1) * block_size;
        }
        blocks.push_back(str.substr(i * block_size, block_size));
```

```
    }
    return blocks;
}

NTRU::NTRU(int _N, int _p, int _q) {
  N = _N;
  p = _p;
  q = _q;
    NTL::ZZ_p::init(NTL::ZZ(q));
    Zx_Ring = NTL::ZZ_pX(NTL::INIT_MONO, N) - 1;
}

std::vector<int> NTRU::cvt_char_to_3_code(char c) {
    std::vector<int> result;
    int ascii_code = static_cast<int>(c);
    if (ascii_code == 0) return std::vector<int> {0};
    int mod = 0;
    while (ascii_code > 0) {
        mod = ascii_code % 3;
        result.push_back(mod);
        ascii_code = ascii_code / 3;
    }
    std::reverse(result.begin(), result.end());
    return result;
}

std::vector<int> NTRU::cvt_string_to_polynomial_coeffs(std::string str) {
    std::vector<int> result;
    const char* str_char = str.c_str();
    for (int i = 0; i < str.length(); i++) {
        std::vector<int> char_3_code = cvt_char_to_3_code(str_char[i]);
        std::vector<int> zeros(6 - char_3_code.size());
        std::fill(zeros.begin(), zeros.end(), 0);
        zeros.insert(zeros.end(), char_3_code.begin(), char_3_code.end());
        result.insert(result.end(), zeros.begin(), zeros.end());
    }
    for (int i = 0; i < result.size(); i++) result[i] -= 1;
    return result;
}

char NTRU::cvt_3_code_to_char(std::vector<int> char_3_code) {
    int char_ascii_code = 0;
    for (int i = 0; i < char_3_code.size(); i++) {
        char_ascii_code += (char_3_code[char_3_code.size() - (i + 1)] * pow(3, i));
    }
    char res = static_cast<char>(char_ascii_code);
    return res;
```

```
}

std::string NTRU::cvt_polynomial_coeffs_to_string(std::vector<int>
    poly_coeffs) {
    std::string res_str = "";
    for (int i = 0; i < poly_coeffs.size(); i++) poly_coeffs[i] += 1;
    int i = 0;
    while (i < poly_coeffs.size()) {
        std::vector<int> vec_slice = slice(poly_coeffs, i, i + 6);
        char c = cvt_3_code_to_char(vec_slice);
        res_str += c;
        i += 6;
    }
    return res_str;
}

std::vector<int> NTRU::random_polynomial_coeffs(int n) {
    std::vector<int> res(n);
    srand((unsigned int)time(NULL));
    const int max = 1;
    const int min = -1;
    for (int i = 0; i < n; i++) {
        res[i] = min + rand() % (max - min + 1);
    }
    return res;
}

NTL::ZZX NTRU::cvt_coeffs_vec_to_ntl_polynomial(std::vector<int>
    coeffs) {
    NTL::ZZX res_poly;
    for (int i = 0; i < coeffs.size(); i++) {
        NTL::SetCoeff(res_poly, i, coeffs[i]);
    }
    return res_poly;
}

std::vector<int> NTRU::cvt_ntl_polynomial_to_coeffs_vec(NTL::ZZX poly
    ) {
    std::vector<int> res_vec;
    for (int i = 0; i <= NTL::deg(poly); i++) {
        long tmp = 0;
        NTL::conv(tmp, poly[i]);
        res_vec.push_back(static_cast<int>(tmp));
    }
    return res_vec;
}

NTL::ZZX NTRU::random_polynomial(int N, int count_positive, int
    count_negative) {

    std::vector<int> poly_coeffs_positive(count_positive);
```

```
    std::fill(poly_coeffs_positive.begin(), poly_coeffs_positive.end(), 1);

    std::vector<int> poly_coeffs_negative(count_negative);
    std::fill(poly_coeffs_negative.begin(), poly_coeffs_negative.end(), -1);

    std::vector<int> poly_coeffs_zero(N - count_positive - count_negative);
    std::fill(poly_coeffs_zero.begin(), poly_coeffs_zero.end(), 0);

    poly_coeffs_positive.insert(poly_coeffs_positive.end(), poly_coeffs_negative.begin(), poly_coeffs_negative.end());
    poly_coeffs_positive.insert(poly_coeffs_positive.end(), poly_coeffs_zero.begin(), poly_coeffs_zero.end());
    std::random_device rd;
    std::mt19937 g(rd());
    std::shuffle(poly_coeffs_positive.begin(), poly_coeffs_positive.end(), g);

    NTL::ZZX res = cvt_coeffs_vec_to_ntl_polynomial(poly_coeffs_positive);

    return res;
}

void NTRU::create_private_keys() {
    int t = N / 3;
    bool flag = true;
    while (flag) {
        f = random_polynomial(N, t + 1, t);
        g = random_polynomial(N, t, t);
        try {
            self_test_invert();
            flag = false;
        }
        catch (NTL::ArithmeticErrorObject e) {
            //std::cout << e.what() << std::endl;
        }
    }
}

void NTRU::create_public_key() {
    // h = f_q * g (mod q)
    NTL::ZZ_pX f_tmp = cvt_ZZX_to_ZZ_pX(f);
    NTL::ZZ_pX f_tmp_inv = invert_poly(f_tmp);
    NTL::ZZ_pX g_tmp = cvt_ZZX_to_ZZ_pX(g);
    h = NTL::MulMod(f_tmp_inv, g_tmp, Zx_Ring);
}

void NTRU::self_test_invert() {
```

```
    NTL::ZZ_p::init(NTL::ZZ(q));
    Zx_Ring = NTL::ZZ_pX(NTL::INIT_MONO, N) - 1;
    NTL::ZZ_pX f_modq = cvt_ZZX_to_ZZ_pX(f);
    NTL::ZZ_pX g_modq = cvt_ZZX_to_ZZ_pX(g);
}

void NTRU::self_test_equals() {
    std::string src_message = "Hello, world!";
    NTL::ZZX message_poly = cvt_coeffs_vec_to_ntl_polynomial(cvt_string_to_polynomial_coeffs(src_message));
    NTL::ZZ_pX encrypted = encrypt(message_poly, true);
    NTL::ZZX decrypted_message_poly = decrypt(encrypted);
    std::string decr_message = cvt_polynomial_coeffs_to_string(cvt_ntl_polynomial_to_coeffs_vec(decrypted_message_poly));
    if (decr_message != src_message) throw NTL::ArithmeticErrorObject("msgs not equal");
}

NTL::ZZ_pX NTRU::cvt_ZZX_to_ZZ_pX(NTL::ZZX poly) {
    NTL::ZZ_pX tmp;
    std::stringstream sstream;
    sstream << poly;
    sstream >> tmp;
    return tmp;
}

NTL::GF2X NTRU::cvt_ZZ_pX_to_GF2X(NTL::ZZ_pX poly) {
    NTL::GF2X tmp;
    std::stringstream sstream;
    sstream << poly;
    sstream >> tmp;
    return tmp;
}

NTL::ZZ_pX NTRU::cvt_GF2X_to_ZZ_pX(NTL::GF2X poly) {
    NTL::ZZ_pX tmp;
    std::stringstream buffer2;
    buffer2 << poly;
    buffer2 >> tmp;
    return tmp;
}

NTL::ZZX NTRU::cvt_ZZ_pX_to_ZZX(NTL::ZZ_pX poly) {
    NTL::ZZ one = NTL::ZZ(1);
    NTL::ZZX tmp;
    NTL::CRT(tmp, one, poly);
    return tmp;
}

NTL::ZZ_pX NTRU::invert_poly(NTL::ZZ_pX poly) {
    int k = 2;
```

```
    NTL::GF2X poly_tmp = cvt_ZZ_pX_to_GF2X(poly);
    NTL::GF2X Ring_tmp = NTL::GF2X(NTL::INIT_MONO, N) - 1;
    NTL::GF2X poly2inv = NTL::InvMod(poly_tmp, Ring_tmp);
    NTL::ZZ_p::init(NTL::ZZ(k));
    NTL::ZZ_pX f_inv = cvt_GF2X_to_ZZ_pX(poly2inv);
    NTL::ZZ_pX Zx_Ring;
    while (k < q) {
        k = k * k;
        NTL::ZZ_p::init(NTL::ZZ(k));
        Zx_Ring = NTL::ZZ_pX(NTL::INIT_MONO, N) - 1;
        f_inv = NTL::MulMod(f_inv, 2 - NTL::MulMod(poly, f_inv,
   Zx_Ring), Zx_Ring);
    }
    NTL::ZZ_p::init(NTL::ZZ(q));
    NTL::ZZ one = NTL::ZZ(1);
    NTL::ZZX tmp;
    NTL::CRT(tmp, one, f_inv);
    return cvt_ZZX_to_ZZ_pX(tmp);
}

NTL::ZZ_pX NTRU::encrypt(NTL::ZZX message, bool check) {
    // e = pr*h+m (mod q)
    int iter_counter = 0;
    while (true) {
        NTL::ZZ_p::init(NTL::ZZ(q));
        Zx_Ring = NTL::ZZ_pX(NTL::INIT_MONO, N) - 1;
        NTL::ZZ_pX r = cvt_ZZX_to_ZZ_pX(random_polynomial(N, N / 3, N
    / 3));
        NTL::ZZ_pX rh = NTL::MulMod(r, h, Zx_Ring);
        NTL::ZZ_pX m = cvt_ZZX_to_ZZ_pX(message);
        NTL::ZZ_pX e = (p * rh) + m;
        if (!check) return e;
        std::string a = cvt_polynomial_coeffs_to_string(
   cvt_ntl_polynomial_to_coeffs_vec(decrypt(e)));
        std::string b = cvt_polynomial_coeffs_to_string(
   cvt_ntl_polynomial_to_coeffs_vec(message));
        if ( a == b || iter_counter > 100) return e;
        iter_counter++;
    }
}

NTL::ZZX NTRU::decrypt(NTL::ZZ_pX encrypted_message) {
    // a = f * e (mod q)
    // m = f_p * a (mod q)
    // fix race condition: tmp ring

    NTL::ZZ_p::init(NTL::ZZ(q));
    NTL::ZZ_pX Zx_Ring_tmp = NTL::ZZ_pX(NTL::INIT_MONO, N) - 1;
    NTL::ZZ_pX f_q = cvt_ZZX_to_ZZ_pX(f);
    NTL::ZZ_pX a_tmp = NTL::MulMod(f_q, encrypted_message,
   Zx_Ring_tmp);
```

```
    NTL::ZZX a = cvt_ZZ_pX_to_ZZX(a_tmp);
    NTL::ZZ_p::init(NTL::ZZ(p));
    Zx_Ring_tmp = NTL::ZZ_pX(NTL::INIT_MONO, N) - 1;
    NTL::ZZ_pX a_p = cvt_ZZX_to_ZZ_pX(a);
    NTL::ZZ_pX f_p = cvt_ZZX_to_ZZ_pX(f);
    NTL::ZZX m = cvt_ZZ_pX_to_ZZX(NTL::MulMod(InvMod(f_p, Zx_Ring_tmp), a_p, Zx_Ring_tmp));
    NTL::ZZ_p::init(NTL::ZZ(q));
    Zx_Ring_tmp = NTL::ZZ_pX(NTL::INIT_MONO, N) - 1;
    return m;
}

void NTRU::save_private_f_to_file(const char* filename) {
    std::ofstream file_output(filename, std::ios_base::out | std::ios_base::trunc);
    if (!file_output.is_open()) { std::cout << "can't open " << filename << std::endl; exit(-1); }
    for (int i = 0; i <= NTL::deg(f); i++) {
        file_output << f[i] << std::endl;
    }
    file_output.close();
}

void NTRU::save_private_g_to_file(const char* filename) {
    std::ofstream file_output(filename, std::ios_base::out | std::ios_base::trunc);
    if (!file_output.is_open()) { std::cout << "can't open " << filename << std::endl; exit(-1); }
    for (int i = 0; i <= NTL::deg(g); i++) {
        file_output << g[i] << std::endl;
    }
    file_output.close();
}

void NTRU::save_public_h_to_file(const char* filename) {
    std::ofstream file_output(filename, std::ios_base::out | std::ios_base::trunc);
    if (!file_output.is_open()) { std::cout << "can't open " << filename << std::endl; exit(-1); }
    for (int i = 0; i <= NTL::deg(h); i++) {
        file_output << h[i] << std::endl;
    }
    file_output.close();
}

void NTRU::save_encrypted_to_file(NTL::ZZ_pX encrypted, const char* filename) {
    std::ofstream file_output(filename, std::ios_base::out | std::ios_base::trunc);
    if (!file_output.is_open()) { std::cout << "can't open " << filename << std::endl; exit(-1); }
```

```
    for (int i = 0; i <= NTL::deg(encrypted); i++) {
        file_output << encrypted[i] << std::endl;
    }
    file_output.close();
}

void NTRU::blocked_save_encrypted_to_file(std::vector<NTL::ZZ_pX> encrypted_blocks, const char* filename) {
    std::ofstream file_output(filename, std::ios_base::out | std::ios_base::trunc);
    if (!file_output.is_open()) { std::cout << "can't open " << filename << std::endl; exit(-1); }
    for (int b = 0; b < encrypted_blocks.size(); b++) {
        for (int i = 0; i <= NTL::deg(encrypted_blocks[b]); i++) {
            file_output << encrypted_blocks[b][i] << " ";
        }
        file_output << std::endl;
    }
    file_output.close();
}

void NTRU::load_private_f_from_file(const char* filename) {
    std::fstream input(filename);
    if (!input.is_open()) { std::cout << "can't open " << filename << std::endl; exit(-1); }
    std::vector<int> coeffs;
    while (!input.eof()) {
        std::string s;
        std::getline(input, s);
        if (!s.empty()) coeffs.push_back(std::stoi(s));
    }
    f = cvt_coeffs_vec_to_ntl_polynomial(coeffs);
}

void NTRU::load_private_g_from_file(const char* filename) {
    std::fstream input(filename);
    if (!input.is_open()) { std::cout << "can't open " << filename << std::endl; exit(-1); }
    std::vector<int> coeffs;
    while (!input.eof()) {
        std::string s;
        std::getline(input, s);
        if (!s.empty()) coeffs.push_back(std::stoi(s));
    }
    g = cvt_coeffs_vec_to_ntl_polynomial(coeffs);
}

void NTRU::load_public_h_from_file(const char* filename) {
    std::fstream input(filename);
    if (!input.is_open()) { std::cout << "can't open " << filename << std::endl; exit(-1); }
```

```
    std::vector<int> coeffs;
    while (!input.eof()) {
        std::string s;
        std::getline(input, s);
        if (!s.empty()) coeffs.push_back(std::stoi(s));
    }
    for (int i = 0; i < coeffs.size(); i++) {
        NTL::SetCoeff(h, i, coeffs[i]);
    }
}

NTL::ZZ_pX NTRU::load_encrypted_from_file(const char* filename) {
    std::fstream input(filename);
    if (!input.is_open()) { std::cout << "can't open " << filename <<
    std::endl; exit(-1); }
    std::vector<int> coeffs;
    while (!input.eof()) {
        std::string s;
        std::getline(input, s);
        if (!s.empty()) coeffs.push_back(std::stoi(s));
    }
    NTL::ZZ_pX encrypted;
    for (int i = 0; i < coeffs.size(); i++) {
        NTL::SetCoeff(encrypted, i, coeffs[i]);
    }
    return encrypted;
}

std::vector<NTL::ZZ_pX> NTRU::blocked_load_encrypted_from_file(const
    char* filename) {
    std::vector <NTL::ZZ_pX> encrypted_blocks;
    std::fstream input(filename);
    if (!input.is_open()) { std::cout << "can't open " << filename <<
    std::endl; exit(-1); }

    // blocks loop
    while (!input.eof()) {
        std::string s;
        std::getline(input, s);
        if (!s.empty()) {
            std::vector<int> block_coeffs;
            std::stringstream ss(s);
            std::string item;
            // coeffs loop
            while (std::getline(ss, item, ' ')) {
                block_coeffs.push_back(std::stoi(item));
            }
            // coeffs to zz_px loop
            NTL::ZZ_pX encrypted_block_tmp;
            for (int i = 0; i < block_coeffs.size(); i++) {
                NTL::SetCoeff(encrypted_block_tmp, i, block_coeffs[i
```

```
]);
            }
            encrypted_blocks.push_back(encrypted_block_tmp);
        }
    }
    return encrypted_blocks;
}

NTL::ZZ_pX NTRU::encrypt_str(std::string str, bool check) {
    std::vector<int> str_coeffs = cvt_string_to_polynomial_coeffs(str);
    NTL::ZZX str_poly = cvt_coeffs_vec_to_ntl_polynomial(str_coeffs);
    NTL::ZZ_pX encrypted_str_poly = encrypt(str_poly, check);
    return encrypted_str_poly;
}

std::string NTRU::decrypt_str(NTL::ZZ_pX encrypted_str_poly) {
    NTL::ZZX str_poly = decrypt(encrypted_str_poly);
    std::vector<int> str_coeffs = cvt_ntl_polynomial_to_coeffs_vec(str_poly);
    std::string str = cvt_polynomial_coeffs_to_string(str_coeffs);
    return str;
}

std::vector<NTL::ZZ_pX> NTRU::blocked_encrypt_str(std::string str, bool check, int block_size) {
    auto src_message_blocks = split_string_to_blocks(str, 12);
    std::vector<NTL::ZZ_pX> encrypted_blocks(src_message_blocks.size());
    char sep = '\\';
#pragma omp parallel for
    for (int i = 0; i < src_message_blocks.size(); i++) {
        encrypted_blocks[i] = encrypt_str(src_message_blocks[i] + sep, true);
    }
    return encrypted_blocks;
}

std::string NTRU::blocked_decrypt_str(std::vector<NTL::ZZ_pX> encrypted_blocks) {
    std::vector<std::string> decrypted_blocks(encrypted_blocks.size());
    char sep = '\\';
#pragma omp parallel for
    for (int i = 0; i < encrypted_blocks.size(); i++) {
        decrypted_blocks[i] = decrypt_str(encrypted_blocks[i]);
        decrypted_blocks[i].erase(std::remove(decrypted_blocks[i].begin(), decrypted_blocks[i].end(), sep), decrypted_blocks[i].end());
    }
    std::string decr_message = std::accumulate(decrypted_blocks.begin
```

```
    (), decrypted_blocks.end(), std::string(""));
    return decr_message;
}

// only for perf tests
std::vector<NTL::ZZ_pX> NTRU::blocked_encrypt_str_seq(std::string str
    , bool check, int block_size) {
    auto src_message_blocks = split_string_to_blocks(str, 12);
    std::vector<NTL::ZZ_pX> encrypted_blocks(src_message_blocks.size
    ());
    char sep = '\\';
    for (int i = 0; i < src_message_blocks.size(); i++) {
        encrypted_blocks[i] = encrypt_str(src_message_blocks[i] + sep
    , true);
    }
    return encrypted_blocks;
}

// only for perf tests
std::string NTRU::blocked_decrypt_str_seq(std::vector<NTL::ZZ_pX>
    encrypted_blocks) {
    std::vector<std::string> decrypted_blocks(encrypted_blocks.size()
    );
    char sep = '\\';
    for (int i = 0; i < encrypted_blocks.size(); i++) {
        decrypted_blocks[i] = decrypt_str(encrypted_blocks[i]);
        decrypted_blocks[i].erase(std::remove(decrypted_blocks[i].
    begin(), decrypted_blocks[i].end(), sep), decrypted_blocks[i].end
    ());
    }
    std::string decr_message = std::accumulate(decrypted_blocks.begin
    (), decrypted_blocks.end(), std::string(""));
    return decr_message;
}

```

Приложение 3. Исходный код интерфейса командной строки main.cpp

```
#include <iostream>
#include <string>
#include <vector>
#include <algorithm>
#include <chrono>
#include "NTRU.h"

```

```
std::string generateRandomString(size_t length) {
    const char* charmap = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789;!&?";
    const size_t charmapLength = strlen(charmap);
    auto generator = [&]() { return charmap[rand() % charmapLength]; };
    std::string result;
    result.reserve(length);
    generate_n(back_inserter(result), length, generator);
    return result;
}

std::string read_file_to_string(std::string filename) {
    std::ifstream t(filename);
    std::string str;
    t.seekg(0, std::ios::end);
    str.reserve(t.tellg());
    t.seekg(0, std::ios::beg);
    str.assign((std::istreambuf_iterator<char>(t)), std::istreambuf_iterator<char>());
    return str;
}

void write_string_to_file(std::string str, std::string filename) {
    std::ofstream file_output(filename, std::ios_base::out | std::ios_base::trunc);
    if (!file_output.is_open()) { std::cout << "can't open " << filename << std::endl; exit(-1); }
    file_output << str;
    file_output.close();
}

void print_help() {
    std::cout << "Args error. Expected:\n\tkeys <f_file> <g_file> <h_file>\n\tenc <input_file> <h_file> <output_file>\n\tdec <input_file> <f_file> <g_file> <output_file>\n";
}

int CLI(int argc, char* argv[]) {
    NTRU ntru(257, 3, 256);
    if (argc < 2) {
        print_help();
        return -1;
    }
    if (std::string(argv[1]) == "keys") {
        std::string key_f_filename = argv[2];
        std::string key_g_filename = argv[3];
        std::string key_h_filename = argv[4];
```

```
        ntru.create_private_keys();
        ntru.create_public_key();
        ntru.save_private_f_to_file(key_f_filename.c_str());
        ntru.save_private_g_to_file(key_g_filename.c_str());
        ntru.save_public_h_to_file(key_h_filename.c_str());
    }
    else if (std::string(argv[1]) == "enc") {
        std::string src_filename = argv[2];
        std::string key_f_filename = argv[3];
        std::string key_g_filename = argv[4];
        std::string key_h_filename = argv[5];
        std::string enc_filename = argv[6];
        ntru.load_public_h_from_file(key_h_filename.c_str());
        ntru.load_private_f_from_file(key_f_filename.c_str());
        ntru.load_private_g_from_file(key_g_filename.c_str());
        std::string src_message = read_file_to_string(src_filename);
        auto encrypted_blocks = ntru.blocked_encrypt_str(src_message,
    true);
        ntru.blocked_save_encrypted_to_file(encrypted_blocks,
   enc_filename.c_str());
    }
    else if (std::string(argv[1]) == "dec") {
        std::string enc_filename = argv[2];
        std::string key_f_filename = argv[3];
        std::string key_g_filename = argv[4];
        std::string dec_filename = argv[5];
        std::vector<NTL::ZZ_pX> encrypted_blocks;
        encrypted_blocks = ntru.blocked_load_encrypted_from_file(
   enc_filename.c_str());
        ntru.load_private_f_from_file(key_f_filename.c_str());
        ntru.load_private_g_from_file(key_g_filename.c_str());
        std::string decr_message = ntru.blocked_decrypt_str(
   encrypted_blocks);
        write_string_to_file(decr_message, dec_filename);
    }
    else {
        print_help();
        return -1;
    }
    return 0;
}

int performance_test() {

    typedef std::chrono::high_resolution_clock Time;
    typedef std::chrono::milliseconds ms;

    std::string key_f_filename = "perf_f.txt";
    std::string key_g_filename = "perf_g.txt";
    std::string key_h_filename = "perf_h.txt";
    std::string src_filename = "perf_src.txt";
```

```
    std::string enc_filename = "perf_enc.txt";
    std::string dec_filename = "perf_dec.txt";

    NTRU ntru(257, 3, 256);

    std::cout << "Create keys performance." << std::endl;
    auto t0 = Time::now();
    ntru.create_private_keys();
    ntru.create_public_key();
    auto t1 = Time::now();
    ntru.save_private_f_to_file(key_f_filename.c_str());
    ntru.save_private_g_to_file(key_g_filename.c_str());
    ntru.save_public_h_to_file(key_h_filename.c_str());
    ms d = std::chrono::duration_cast<ms>(t1 - t0);
    std::cout << "\t time: " << d.count() << " ms" << std::endl;

    std::cout << "Encrypt performance." << std::endl;
    ntru.load_public_h_from_file(key_h_filename.c_str());
    ntru.load_private_f_from_file(key_f_filename.c_str());
    ntru.load_private_g_from_file(key_g_filename.c_str());
    std::string src_message = read_file_to_string(src_filename);
    t0 = Time::now();
    auto encrypted_blocks = ntru.blocked_encrypt_str_seq(src_message, true);
    t1 = Time::now();
    ntru.blocked_save_encrypted_to_file(encrypted_blocks, enc_filename.c_str());
    ms d_seq = std::chrono::duration_cast<ms>(t1 - t0);
    std::cout << "\t time seq: " << d_seq.count() << " ms" << std::endl;
    t0 = Time::now();
    encrypted_blocks = ntru.blocked_encrypt_str(src_message, true);
    t1 = Time::now();
    ntru.blocked_save_encrypted_to_file(encrypted_blocks, enc_filename.c_str());
    ms d_par = std::chrono::duration_cast<ms>(t1 - t0);
    std::cout << "\t time par: " << d_par.count() << " ms" << std::endl;

    std::cout << "Decrypt performance." << std::endl;
    ntru.load_private_f_from_file(key_f_filename.c_str());
    ntru.load_private_g_from_file(key_g_filename.c_str());
    encrypted_blocks = ntru.blocked_load_encrypted_from_file(enc_filename.c_str());
    t0 = Time::now();
    std::string decr_message = ntru.blocked_decrypt_str_seq(encrypted_blocks);
    t1 = Time::now();
    write_string_to_file(decr_message, dec_filename);
    d_seq = std::chrono::duration_cast<ms>(t1 - t0);
    std::cout << "\t time seq: " << d_seq.count() << " ms" << std::
```

```
    endl;
    encrypted_blocks = ntru.blocked_load_encrypted_from_file(enc_filename.c_str());
    t0 = Time::now();
    decr_message = ntru.blocked_decrypt_str(encrypted_blocks);
    t1 = Time::now();
    write_string_to_file(decr_message, dec_filename);
    d_par = std::chrono::duration_cast<ms>(t1 - t0);
    std::cout << "\t time par: " << d_par.count() << " ms" << std::endl;
}


int main(int argc, char* argv[]) {
    return(CLI(argc, argv));
    // return(performance_test());
}

```

Приложение 4. Исходный графического интерфейса mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "ntru.h"
#include <string>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private slots:
    void on_bt_select_source_clicked();

    void on_bt_select_encrypted_clicked();

```

```
    void on_bt_select_key_h_clicked();

    void on_bt_select_key_f_clicked();

    void on_bt_select_key_g_clicked();

    void on_bt_create_keys_clicked();

    void on_bt_encrypt_clicked();

    void on_bt_decrypt_clicked();

    void on_bt_show_source_clicked();

    void on_bt_show_encrypted_clicked();

    void on_bt_show_decrypted_clicked();

    void on_bt_show_key_h_clicked();

    void on_bt_show_key_f_clicked();

    void on_bt_show_key_g_clicked();

    void on_bt_select_decrypted_clicked();

private:
    Ui::MainWindow *ui;
    NTRU ntru = NTRU(257, 3, 256);
    std::string src_filename = "";
    std::string enc_filename = "";
    std::string dec_filename = "";
    std::string key_h_filename = "";
    std::string key_f_filename = "";
    std::string key_g_filename = "";
};
#endif // MAINWINDOW_H

```

Приложение 5. Исходный графического интерфейса mainwindow.cpp

```
#include "mainwindow.h"
#include "./ui_mainwindow.h"
#include <QFileDialog>
#include <iostream>
```

```

std::string read_file_to_string(std::string filename) {
    std::ifstream t(filename);
    std::string str;
    t.seekg(0, std::ios::end);
    str.reserve(t.tellg());
    t.seekg(0, std::ios::beg);
    str.assign((std::istreambuf_iterator<char>(t)), std::istreambuf_iterator<char>());
    return str;
}

void write_string_to_file(std::string str, std::string filename) {
    std::ofstream file_output(filename, std::ios_base::out | std::ios_base::trunc);
    if (!file_output.is_open()) { std::cout << "can't open " << filename << std::endl; exit(-1); }
    file_output << str;
    file_output.close();
}

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}

MainWindow::~MainWindow()
{
    delete ui;
}


void MainWindow::on_bt_select_source_clicked()
{
    src_filename = QFileDialog::getOpenFileName(this, tr("Source file")).toStdString();
    std::cout << src_filename << std::endl;
}


void MainWindow::on_bt_select_encrypted_clicked()
{
    enc_filename = QFileDialog::getOpenFileName(this, tr("Encrypted file")).toStdString();
    std::cout << enc_filename << std::endl;
}

void MainWindow::on_bt_select_decrypted_clicked()
{
```

```
    dec_filename = QFileDialog::getOpenFileName(this, tr("Decrypted file")).toStdString();
    std::cout << dec_filename << std::endl;
}

void MainWindow::on_bt_select_key_h_clicked()
{
    key_h_filename = QFileDialog::getOpenFileName(this, tr("Public Key h file")).toStdString();
    std::cout << key_h_filename << std::endl;
}


void MainWindow::on_bt_select_key_f_clicked()
{
    key_f_filename = QFileDialog::getOpenFileName(this, tr("Private Key f file")).toStdString();
    std::cout << key_f_filename << std::endl;
}


void MainWindow::on_bt_select_key_g_clicked()
{
    key_g_filename = QFileDialog::getOpenFileName(this, tr("Private Key g file")).toStdString();
    std::cout << key_g_filename << std::endl;
}


void MainWindow::on_bt_create_keys_clicked()
{
    ntru.create_private_keys();
    ntru.create_public_key();
    ntru.save_private_f_to_file(key_f_filename.c_str());
    ntru.save_private_g_to_file(key_g_filename.c_str());
    ntru.save_public_h_to_file(key_h_filename.c_str());
}


void MainWindow::on_bt_encrypt_clicked()
{
    ntru.load_public_h_from_file(key_h_filename.c_str());
    ntru.load_private_f_from_file(key_f_filename.c_str());
    ntru.load_private_g_from_file(key_g_filename.c_str());
    std::string src_message = read_file_to_string(src_filename);
    std::vector<NTL::ZZ_pX> encrypted_blocks = ntru.blocked_encrypt_str(src_message, true);
    ntru.blocked_save_encrypted_to_file(encrypted_blocks, enc_filename.c_str());
}

```

```

void MainWindow::on_bt_decrypt_clicked()
{
    std::vector<NTL::ZZ_pX> encrypted_blocks;
    encrypted_blocks = ntru.blocked_load_encrypted_from_file(
  enc_filename.c_str());
    ntru.load_private_f_from_file(key_f_filename.c_str());
    ntru.load_private_g_from_file(key_g_filename.c_str());
    std::string decr_message = ntru.blocked_decrypt_str(
  encrypted_blocks);
    write_string_to_file(decr_message, dec_filename);
}


void MainWindow::on_bt_show_source_clicked()
{
    std::string src = read_file_to_string(src_filename);
    ui->plainTextEdit->setPlainText(QString::fromStdString(src));
}


void MainWindow::on_bt_show_encrypted_clicked()
{
    std::string enc = read_file_to_string(enc_filename);
    ui->plainTextEdit->setPlainText(QString::fromStdString(enc));
}


void MainWindow::on_bt_show_decrypted_clicked()
{
    std::string dec = read_file_to_string(dec_filename);
    ui->plainTextEdit->setPlainText(QString::fromStdString(dec));
}


void MainWindow::on_bt_show_key_h_clicked()
{
    std::string key_h = read_file_to_string(key_h_filename);
    ui->plainTextEdit->setPlainText(QString::fromStdString(key_h));
}


void MainWindow::on_bt_show_key_f_clicked()
{
    std::string key_f = read_file_to_string(key_f_filename);
    ui->plainTextEdit->setPlainText(QString::fromStdString(key_f));
}


void MainWindow::on_bt_show_key_g_clicked()
{
```

```
    std::string key_g = read_file_to_string(key_g_filename);
    ui->plainTextEdit->setPlainText(QString::fromStdString(key_g));
}

```

Приложение 6. Исходный графического интерфейса main.cpp

```
#include "mainwindow.h"

#include <QApplication>


int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}

```