МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра систем информатики

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

Усольцева Мария Александровна

Защищенный калькулятор. Разработка серверного компонента

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Руководитель

Кренделев С. Ф.
(фамилия , И., О.)
канд. физ.-мат. наук, доцент НГУ
(уч.степень, уч.звание)

……..
(подпись, дата)

Автор

Усольцева М. А.
(фамилия , И., О.)
ФИТ, 9201
(факультет, группа)

……..
(подпись, дата)

Новосибирск, 2013 г.

МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ» (НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Кафедра систем информатики

УТВЕРЖДАЮ

Зав. Кафедрой Лаврентьев М. М.
(фамилия, И., О.)

…….
(подпись, дата)

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ БАКАЛАВРА

Студентке Усольцевой Марии Александровне

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Тема: «Защищенный калькулятор. Разработка серверного компонента»

Исходные данные (или цель работы): разработка двух схем полностью гомоморфного шифрования на основе полиномов с коэффициентами из кольца целых чисел, исследование их свойств, реализация разработанных схем в виде серверного компонента приложения «Защищенный калькулятор», анализ производительности схем.

Структурные части работы: разработка математической модели схем в виде алгоритмов шифрования, дешифрования, генерации секретных ключей и вычислений над зашифрованными данными, анализ криптографической стойкости данных схем к различным видам атак, анализ сложности операций, реализация схем в виде серверного компонента приложения «Защищенный калькулятор», статистические данные о производительности схем.

Содержание

Введение

Облачные вычисления – одно из наиболее быстро развивающихся направлений в современных информационных технологиях, приложения для использования в облачных сервисах являются приоритетными проектами у ведущих мировых ИТ-компаний. Этот подход позволяет организовать динамическое предоставление услуг, что дает возможность пользователям производить оплату по факту использования ресурсов и регулировать их объем в зависимости от реальных потребностей без долгосрочных обязательств.

Основной причиной развития облачных сервисов является возможность снижения расходов компаний и частных лиц на поддержание собственной ИТ-инфраструктуры за счёт передачи этой работы провайдеру облачного сервиса.

Облачные вычисления являются эффективным инструментом повышения прибыли и расширения каналов продаж для независимых производителей программного обеспечения ISV *(Independent Software Vendor*), операторов связи и VAR-посредников в форме SaaS *(Software as a service — программное обеспечение как услуга)*.

В 2011 году Национальным институтом стандартом и технологий США (NIST) было дано «окончательное» определения понятия «облачные вычисления» [1].

Определение 1. Облачные вычисления (англ. cloud computing) — это модель обеспечения повсеместного и удобного сетевого доступа по требованию к общему пулу (англ. pool) конфигурируемых вычислительных ресурсов (например, сетям передачи данных, серверам, устройствам хранения данных, приложениям и сервисам — как вместе, так и по отдельности), которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами и/или обращениями к провайдеру.

К ключевым характеристикам облачных вычислений относятся:

Самообслуживание по запросу. Потребитель, в одностороннем порядке, заказывает компьютерные ресурсы, такие, как серверное время или объём хранилища, и получает их автоматически, без взаимодействия с персоналом сервис-провайдера.

Доступ через сеть из любой точки. Ресурсы и услуги доставляются через сеть, доступ к ним осуществляется стандартным способом, с использованием толстых и тонких клиентов (телефоны, ноутбуки, PDA).

Комбинирование ресурсов. Компьютерные ресурсы провайдера группируются для обслуживания нескольких потребителей, используя коммунальную модель. Физические и виртуальные ресурсы добавляются и удаляются в соответствии с потребностями клиентов.

Существует понятие независимости от местоположения, так как клиент не знает и не может контролировать местонахождение конкретного ресурса. Однако для высокого уровня абстракции (страна, регион, датацентр) клиенту может быть предоставлен выбор. Примерами ресурсов являются хранилище данных, обработка данных, память (объём), ширина канала, виртуальная машина.

Высокая (быстрая) эластичность. Объёмы предоставляемых ресурсов могут быть быстро увеличены и быстро сокращены, в некоторых случаях – автоматически. Для потребителя это создаёт иллюзию бесконечности предоставляемых ресурсов, которые могут быть доступны в любом количестве, в любое время.

Измеримость сервиса. Облачные системы автоматически контролируют и оптимизируют использование ресурсов за счёт качества измеримости сервиса на некотором уровне абстракции, соответствующему типу сервиса (хранилище данных, обработка данных, передача данных, количество активных учётных записей пользователей). Использование ресурсов подвергается мониторингу, контролю и обработке, что даёт прозрачность в понимании использования сервиса провайдерами и клиентами. Типовая бизнес-модель – оплата по факту использования ресурса.

Одной из нерешенных проблем облачных вычислений является безопасность хранимых данных. Прежде чем потребитель доверит свои данные облачному сервису, необходимо гарантировать сохранение их конфиденциальности и исключить или минимизировать вероятность несанкционированного доступа к информации, в том числе со стороны владельца облачного сервиса.

При использовании сервера в качестве хранилища данных, особых проблем не возникает – данные могут быть зашифрованы ключом клиента и тем самым защищены от злоумышленника. Однако если необходимо проводить вычисления с данными клиента, то возникает необходимость их дешифрования, что нарушает конфиденциальность обрабатываемой информации.

С практическим внедрением метода полностью гомоморфного шифрования, позволяющего производить операции над открытым текстом путем операций над шифротекстом, эта проблема в значительной степени будет решена. Полностью гомоморфная криптографическая система является гомоморфной относительно операций умножения и сложения одновременно, то есть, она сохраняет структуру колец открытых текстов. Используя такую систему, будет возможно создавать программы, которые могут быть запущены на шифровании ввода, чтобы произвести шифрование вывода. Так как

такие программы никогда не дешифрует свой ввод, они могут быть выполнены недостоверной стороной, не показывая свой ввод и внутреннее состояние.

Алгоритм гомоморфного шифрования предложили еще основатели современной криптографии – Рональд Ривест (Ronald Rivest), Леонард Адлеман (Leonard Adleman) и Майкл Дертузо (Michael Dertouzos). Тем не менее, найти полное решение теоретической задачи полностью гомоморфного шифрования до сих пор не удавалось – известны были лишь частные решения. К таким решениям, к примеру, относится алгоритм RSA, гомоморфный только относительно умножения.

Впервые схема полностью гомоморфного шифрования была предложена исследователем из IBM Крейгом Гентри в 2009 году. Однако схема до сих пор не является достаточно удобной для практического применения. При операциях сложения и умножения размер зашифрованных данных растет экспоненциально, а для его редуцирования требуется трудоемкая в вычислительном плане операция перешифровки данных. По оценке самого Гентри, например, обработка поискового запроса в Google в случае, если текст зашифрован, потребует примерно в триллион раз больше вычислений.

Таким образом, создание гомоморфной криптографической системы с приемлемой скоростью работы является открытой проблемой, следовательно, исследования в данной области являются актуальными.

Глава 1. Постановка задачи

1.1 Цели и задачи исследования

Целью данного исследования является разработка двух схем полностью гомоморфного шифрования, использующих гомоморфизмы колец полиномов с целыми коэффициентами $\mathbb{Z}[x]$ и реализующих операции сложения, умножения, вычитания, деления над целыми числами.

В рамках работы были поставлены следующие задачи:

- исследование существующей схемы полностью гомоморфного шифрования, предложенной Крейгом Гентри;
- разработка математической модели схем в виде алгоритмов шифрования, дешифрования, генерации секретных ключей и вычислений над зашифрованными данными;
- анализ криптографической стойкости разработанных схем к различным атакам;
- реализация серверного компонента приложения «Защищенный калькулятор», демонстрирующего возможности разработанных схем;
- анализ производительности разработанных схем.

1.2. Математическая формулировка

В основе данного исследования лежит набор следующих математических понятий.

Определение 2. Гомоморфизм алгебраических систем. Пусть даны две однотипные алгебраические системы:

$$U = < S;\ f_1, \dots, f_k;\ p_1, \dots, p_l;\ s_1, \dots, s_m >,$$

$$C = < S';\ f_1', \dots, f_k';\ p_1', \dots, p_k';\ s_1', \dots, s_m' >,$$

где S и S'– множества; $f_1, \dots, f_k$, $f_1', \dots, f_k'$ – функции, $p_1, \dots, p_l$, $p_1', \dots, p_k'$ – предикаты, определенные на множествах S и S'соответственно, $\{s_1, \dots, s_m\} \subset S, \{s_1', \dots, s_m'\} \subset S'$– известные константы.

Отображение $\phi : S \to S'$ называется ***гомоморфизмом*** алгебраической системы U в C, если выполняются следующие условия:

1) $\forall i \in \{1, \dots, k\}$: $\phi(f_i(a_1, \dots, a_n)) = f_i'(\phi(a_1), \dots, \phi(a_n))$

2) $\forall i \in \{1, \dots, l\}: p_i(a_1, \dots, a_n) = p'_i(\phi(a_1), \dots, \phi(a_n))$

3) $\forall i \in \{1, \dots, m\}: \phi(s_i) = s'_i$

В работе [2] было введено понятие ***privacy homomorphism***, которое накладывает следующие условия на отображение ϕ:

1) $\phi : S \to S'$ является шифрующей функцией;
   существует обратная функция $\phi^{-1} : S' \to S$, являющаяся дешифрующей;
2) функции ϕ и ϕ^{-1} легко вычислимы;
3) операции f'_i и предикаты p'_i также эффективно вычислимы;
4) представление зашифрованного значения $\phi(d_i)$, где $d_i \in S$, не должно занимать намного больше места, чем представление незашифрованного значения d_i;
5) знания $\phi(d_i)$ для большого количества d_i недостаточно для восстановления ϕ (атака с известным шифротекстом);
6) знания нескольких пар $(d_i, \phi(d_i))$ недостаточно для восстановления ϕ (атака с подобранным открытым текстом);
7) знание алгоритмов вычисления операций f'_i и предикатов p'_i не достаточно для построения алгоритма вычисления ϕ.

Определение 3. Гомоморфное шифрование. Будем говорить, что для алгебраических систем U и C из определения 2 можно построить *гомоморфное шифрование*, если:

1) существует достаточно большое количество функций ϕ, удовлетворяющих определению *privacy homomorphism*, причём такие функции достаточно легко строить;
2) число операций $k > 0$.

Таким образом, *гомоморфное шифрование* является криптографической системой, которая позволяет проводить определенные математические действия с открытым текстом путем операций с зашифрованным текстом.

Определение 4. Полностью гомоморфное шифрование. *Полностью гомоморфное шифрование* – это гомоморфное шифрование, где C и U – кольца, причём среди операций f_i существуют операции $+$ и $\times$.

1.3. Существующие решения

В 2009 году Крэйг Гентри построил схему, которую назвал *«Fully Homomorphic Encryption based on Ideal Lattices»* [3]. Она доказывает возможность построения гомоморфного шифрования. Рассмотрим суть данной работы.

Генерация ключей. Выбирается произвольное нечетное целое число p с количеством бит λ^2, оно будет иметь вид $p = 2k + 1$. Данное число p является секретным параметром.

Шифрование. Пусть требуется зашифровать бит $m \in \{0,1\}$. Для этого построим число z по правилу $z = 2r + m$, где r произвольное целое λ-битовое число. Это означает, что $z = m \; mod \; 2$. Шифрование заключается в том, что всякому числу m сопоставляется число $c = pq + z$, где q произвольное целое λ^5-битовое число. Следовательно, $c = 2r + m + (2k + 1)q = 2(r + kq) + m + q$. Число c отправляется на вычисления. Заметим, что $c \; mod \; 2 = (m + q) \; mod \; 2$. Поэтому возможно только узнать четность выхода из шифрования.

Расшифровка. Процедура расшифровки выглядит следующим образом. Пусть даны числа c, p, q, где p, q известны. Прежде всего, вычисляется $c \; mod \; p = (z + pq) \; mod \; p = z \; mod \; p + pq \; mod \; p = z \; mod \; p = (2r + m) \; mod \; p = 2(r \; mod \; p) + m \; mod \; p$. После чего, вычисляется $(c \; mod \; p) \; mod \; 2 = \big(2(r \; mod \; p)\big) \; mod \; 2 + (m \; mod \; p) \; mod \; 2 = \; m \; mod \; 2 = m$.

Число $c \; mod \; p$ называется «шумом». Возможные значения $c \; mod \; p$ лежат в интервале $(-p/2, \; p/2)$.

Данное шифрование является ***гомоморфным*** относительно операций сложения и умножения. Рассмотрим пару чисел $m_1, m_2 \in \{0,1\}$. Сопоставим им $z_1 = 2r_1 + m_1$, $z_2 = 2r_2 + m_2$. Выберем секретный ключ $p = 2k + 1$. Получим $Enc(m_1) = \; c_1 = z_1 + pq_1$, $Enc(m_2) = \; c_2 = z_2 + pq_2$ – шифротекст для m_1 и m_2 соответственно.

Тогда ***операция сложения*** над зашифрованными числами будет выглядеть следующим образом:

$$Enc(m_1) + Enc(m_2) = \; c_1 + c_2 = z_1 + z_2 + p(q_1 + q_2) =$$
$$= 2r_1 + m_1 + 2r_2 + m_2 + p(q_1 + q_2) = 2(r_1 + r_2) + m_1 + m_2 + p(q_1 + q_2);$$

операция умножения:

$$Enc(m_1) \times Enc(m_2) = c_1c_2 = z_1z_2 + p(z_1q_2 + z_2q_1) + p^2q_1q_2 =$$
$$= (2r_1 + m_1)(2r_2 + m_2) + p(z_1q_2 + z_2q_1) + p^2q_1q_2 =$$
$$= 4r_1r_2 + 2(r_1m_2 + r_2m_1) + m_1m_2 + p(z_1q_2 + z_2q_1) + p^2q_1q_2.$$

При ***дешифровании*** имеем:

$$Dec\big(Enc(m_1) + Enc(m_2)\big) = (c_1 + c_2)\ mod\ p\ mod\ 2 = m_1 + m_2;$$
$$Dec\big(Enc(m_1) \times Enc(m_2)\big) = c_1c_2\ mod\ p\ mod\ 2 = m_1m_2.$$

Недостатком данной схемы является то, что выполнение вычислений ведет к накоплению ошибки и после того, как она превысит секретный ключ p, расшифровать число становится невозможным.

Одним из вариантов решения данной проблемы является перешифрование данных после некоторого количества операций. Однако такой вариант снижает производительность вычислений и требует постоянного доступа к хранилищу закрытого ключа.

Таким образом, данная схема до сих пор не является достаточно удобной для практического применения.

1.4. Технические требования

В рамках исследования требуется создать серверную часть приложения «Защищенный калькулятор», которое должно демонстрировать возможности разработанной схемы полностью гомоморфного шифрования.

В качестве языка для реализации серверной части приложения был выбран язык Java. Причина такого выбора являются:

1) Независимость от платформы, на которой выполняются программы. Java работает на большинстве аппаратных платформ и операционных систем крупных производителей.
2) Наличие двух основных библиотек для создания графических интерфейсов пользователя (Graphics User Interface). Это Abstract Windows Toolkit (AWT) и Swing. Было решено использовать Swing, так как данная библиотека является более новой и обладает более широкими возможностями.
3) Наличие встроенной длинной арифметики в стандартной библиотеке java.math (классы BigDecimal, BigInteger).

Серверная часть приложения должна:

- принимать от клиентской части приложения зашифрованные данные и функцию от этих данных, заданную пользователем;
- применять данную функцию к зашифрованным данным;
- отправлять результат вычислений клиентской части приложения;
- обрабатывать запросы клиентов параллельно.

Также в рамках данного исследования для анализа производительности требуется разработать аналог криптографической библиотеки на языке C++ с использованием арифметической библиотеки GMP.

Данный подход позволит использовать криптографическую библиотеку для других приложений и задач.

1.5. Ожидаемые результаты

В результате данной работы необходимо получить:

1) описание схем полностью гомоморфного шифрования и алгоритмов, реализующих данные схемы;
2) предварительное заключение об устойчивости к криптографическим атакам;
3) экспериментальную реализацию схем в виде серверного компонента приложения;
4) статистические данные о производительности разработанных схем.

Глава 2. Схемы полностью гомоморфного шифрования

2.1. Математическое обоснование схемы 1

Операции над многочленами

Рассмотрим первую подзадачу, необходимую для построения полностью гомоморфного шифрования – установление соответствия между кольцами целых чисел и полиномов, а также операциями сложения и умножения над ними.

Пусть даны два полинома $a(x), b(x) \in \mathbb{Z}[x]$, такие, что

$$a(x) = a_0 + a_1 x + \cdots + a_n x^n,$$
$$b(x) = b_0 + b_1 x + \cdots + b_n x^n,$$

где $n > 0$. Без ограничения общности можно считать, что данные полиномы имеют одинаковые степени, так как в ином случае полиному меньшей степени можно дописать необходимое количество старших членов с нулевыми коэффициентами.

Операции сложения и умножения в кольце полиномов определяются следующим образом.

$$a(x) + b(x) = a_0 + b_0 + (a_1 + b_1)x + \cdots + (a_n + b_n)x^n,$$
$$a(x) \times b(x) = a_0 b_0 + (a_0 b_1 + b_0 a_1)x + \cdots + a_n b_n x^{2n}.$$

Следовательно, свободный член результирующего полинома будет равен сумме и произведению свободных членов исходных полиномов соответственно.

Построение отображения из кольца $\mathbb{Z}$ в кольцо полиномов $\mathbb{Z}[x]$

Далее, рассмотрим задачу построения отображения из кольца целых чисел $\mathbb{Z}$ в кольцо полиномов $\mathbb{Z}[x]$. В качестве ее решения рассмотрим следующий алгоритм.

Алгоритм 1. Сопоставление полинома целому числу.

1) Пусть $z \in \mathbb{Z}$ – некоторое целое число, которому требуется сопоставить полином. Пусть задано число $n > 0$ – степень полинома, число $x_0 = \frac{p}{q}$, $x_0 \in \mathbb{Q}$.
2) Построим полином $f(x) = a_0 + a_1 x + \cdots + a_n x^n$ такой, что коэффициенты $a_0, a_1, \ldots a_n \in \mathbb{Z}$ выбираются случайным образом.
3) Вычислим $f(x_0) = f\left(\frac{p}{q}\right) = a_0 + a_1\left(\frac{p}{q}\right) + \cdots + a_n\left(\frac{p}{q}\right)^n$. Откуда получаем

   $q^n f\left(\frac{p}{q}\right) = q^n a_0 + q^{n-1} p a_1 + \cdots + p^n a_n$, где $q^n f\left(\frac{p}{q}\right) \in \mathbb{Z}$.

4) $g_z(x) = q^n f(x) - q^n f\left(\frac{p}{q}\right) + z$ – полином, соответствующий числу z.

Алгоритм 2. Сопоставление целого числа полиному (обратный алгоритм для алгоритма 1).

1) Пусть дан полином $g_z(x)$, полученный в результате выполнения алгоритма.
2) Тогда $g_z(x_0) = q^n f(x_0) - q^n f\left(\frac{p}{q}\right) + z = z$ – исходное число.

Утверждение 1. Пусть $h : \mathbb{Z} \to \mathbb{Z}[x]$ – отображение, построенное согласно алгоритму 1. Тогда данное отображение является разнозначным.

Доказательство. Доказательство следует очевидным образом из случайного выбора коэффициентов. ■

Построение гомоморфизма из кольца $\mathbb{Z}$ в кольцо полиномов $\mathbb{Z}[x]$

Рассмотрим задачу построения гомоморфизма из кольца Z в кольцо Z[x].

Определение 5. Гомоморфизм колец. Пусть $f : A \to B$, где A и B – кольца со сложением, умножением и единицей. Тогда f – гомоморфизм колец, если:

$$f(a +_A b) = f(a) +_B f(b) \quad (1)$$
$$f(a \times_A b) = f(a) \times_B f(b) \quad (2)$$
$$f(0_A) = 0_B \quad (3)$$
$$f(1_A) = 1_B \quad (4)$$

Теорема 1. Пусть дано кольцо целых чисел $\mathbb{Z}$, кольцо полиномов $\mathbb{Z}[x]$. Тогда отображение $P(x)$, построенное согласно алгоритму, является гомоморфизмом $P : \mathbb{Z} \to \mathbb{Z}[x]$.

Доказательство. Пусть $z_1, z_2 \in \mathbb{Z}$. Покажем, что $P(z_1 +_Z z_2) = P(z_1) +_{Z[x]} P(z_2)$.

$$\begin{aligned} P(z_1) +_{Z[x]} P(z_2) &= q^n f_1(x) - q^n f_1\left(\frac{p}{q}\right) + z_1 + q^n f_2(x) - q^n f_2\left(\frac{p}{q}\right) + z_2 \\ &= q^n a_0 + q^n a_1 x + \cdots + q^n a_n x^n - (q^n a_0 + q^{n-1} p a_1 + \cdots + p^n a_n) + z_1 \\ &+ q^n b_0 + q^n b_1 x + \cdots + q^n b_n x^n - (q^n b_0 + q^{n-1} p b_1 + \cdots + p^n b_n) + z_2 \\ &= (a_1 + b_1)(q^n x - q^{n-1} p) + \cdots + (a_n + b_n)(q^n x - p^n) + z_1 + z_2 \\ &= q^n c_0 + q^n c_1 x + \cdots + q^n c_n x^n - (q^n c_0 + q^{n-1} p c_1 + \cdots + p^n c_n) + z_1 + z_2 \\ &= P(z_1 +_Z z_2), \end{aligned}$$

где $c_i = a_i + b_i$, $i \in \{0 \ldots n\}$.

Свойство (2) доказывается аналогично. Очевидно, что отображение P переводит 1 в 1 и 0 в 0, что соответствует свойствам (3) и (4). Следовательно, отображение P является гомоморфизмом. ■

Далее будет рассмотрена схема 1 гомоморфного шифрования, основанная на вышеизложенном теоретическом материале.

2.2. Схема 1

Генерация ключей. Пусть x_0 – закрытый ключ данного шифра, $x_0 \in \mathbb{Q}$, т.е. $x_0 = \frac{p}{q}$, где $p \in \mathbb{Z}, q \in \mathbb{N}, p, q$ – случайные числа.

Шифрование. Алгоритм шифрования $Enc : \mathbb{Z} \to \mathbb{Z}[x]$ выглядит следующим образом.

1) Пусть $z \in \mathbb{Z}$ – число, которое требуется зашифровать.
2) Построим полином $f(x) = a_0 + a_1 x + \cdots + a_n x^n$ такой, что $n > 0$ и коэффициенты $a_0, a_1, \ldots a_n \in \mathbb{Z}$ выбираются случайным образом.
3) Вычислим $f(x_0) = f\left(\frac{p}{q}\right) = a_0 + a_1\left(\frac{p}{q}\right) + \cdots + a_n\left(\frac{p}{q}\right)^n$. Откуда получаем
   $q^n f\left(\frac{p}{q}\right) = q^n a_0 + q^{n-1} p a_1 + \cdots + p^n a_n$, где $q^n f\left(\frac{p}{q}\right) \in \mathbb{Z}$.
4) Построим шифрующий полином для z следующим образом:
   $g_z(x) = q^n f(x) - q^n f\left(\frac{p}{q}\right) + z \in \mathbb{Z}$.

Расшифровка. Рассмотрим алгоритм расшифровки $Dec : \mathbb{Z}[x] \to \mathbb{Z}$.

1) Пусть полином $g''_{z'}(x)$ – зашифрованные данные, x_0 – закрытый ключ.
2) Для расшифровки необходимо вычислить $g''_{z'}(x)$ в точке x_0.
3) Тогда $z' = g''_{z'}(x_0) \in \mathbb{Z}$ – расшифрованные данные.

Согласно теории рассмотренной в пункте 2.1. данное шифрование является гомоморфным, то есть для $z_1, z_2 \in \mathbb{Z}$ выполняются следующие свойства:

$$z_1 + z_2 = Dec(Enc(z_1) + Enc(z_2))$$
$$z_1 \times z_2 = Dec(Enc(z_1) \times Enc(z_2))$$

В данной криптографической схеме степень n полинома $g_z(x)$ и его коэффициенты является общедоступными параметрами.

2.3. Математическое обоснование схемы 2

Построение отображения из кольца $\mathbb{Z}$ в кольцо полиномов $\mathbb{Z}[x]$

Теорема 2. Китайская теорема об остатках для полиномов. Пусть $\mathbb{K}$ – кольцо и $u_1(x), \dots, u_r(x)$ – попарно взаимно простые многочлены из $\mathbb{K}[x]$. Для любого набора $a_1(x), \dots, a_r(x)$ многочленов из $\mathbb{K}[x]$ существует многочлен $c(x)$, такой, что $c(x) \equiv a_i(x)\big(mod\ u_i(x)\big)$ для любого $i = 1, \dots, r$. Условием $\deg c(x) < \sum_{i=1}^{r} \deg u_i(x)$ многочлен $c(x)$ определяется однозначно.

Алгоритм 3. Сопоставление полинома целому числу.

1) Пусть $z \in \mathbb{Z}$ – некоторое целое число, которому требуется сопоставить полином. Пусть задано число $n > 0$ – степень полинома, алгебраическое число $x_0 \in \mathbb{A}$.
2) Пусть дан неприводимый над полем $\mathbb{Z}$ полином $u(x)$, такой, что $u(x_0) = 0$.
3) Сопоставим числу z полином $g_z(x)$ такой, что $z = g_z(x)\ mod\ u(x)$, т.е. $g_z(x) = s(x)u(x) + z$, где $s(x)$ – полином с произвольными коэффициентами. Это можно сделать согласно Китайской теореме об остатках для полиномов (теорема 2).

Алгоритм 4. Сопоставление целого числа полиному (обратный алгоритм для алгоритма 2).

1) Пусть дан полином $g_z(x)$, полученный в результате выполнения алгоритма.
2) Тогда $g_z(x_0) = s(x_0)u(x_0) + z = z$ – исходное число.

Очевидно, что отображение, построенное согласно алгоритму 3, также является разнозначным.

Построение гомоморфизма из кольца $\mathbb{Z}$ в кольцо полиномов $\mathbb{Z}[x]$

Теорема 3. Пусть дано кольцо целых чисел $\mathbb{Z}$, кольцо полиномов $\mathbb{Z}[x]$. Тогда отображение $P(x)$, построенное согласно алгоритму 3, является гомоморфизмом $P : \mathbb{Z} \to \mathbb{Z}[x]$.

Доказательство. Пусть $z_1, z_2 \in \mathbb{Z}$. Покажем, что $P(z_1 +_Z z_2) = P(z_1) +_{Z[x]} P(z_2)$.

$P(z_1) +_{Z[x]} P(z_2) = s_1(x)u(x) + z_1 + s_2(x)u(x) + z_2 = \big(s_1(x) + s_2(x)\big)u(x) + z_1 + z_2 = P(z_1 +_Z z_2)$.

Остальные свойства из определения 5 доказываются аналогично. ■

Определение 6. Замена переменных в кольце полиномов. Пусть $a(x) = a_m x^m + \dots + a_1 x + a_0$, $p(x) = p_k x^k + \dots + p_1 x + p_0 \in \mathbb{K}[x]$ – полиномы, тогда композиция полиномов

$a'(x) = a\big(p(x)\big) = a_m(p_k x^k + \cdots + p_1 x + p_0)^m + \cdots + a_1(p_k x^k + \cdots + p_1 x + p_0) + a_0$ также является полиномом из кольца $\mathbb{K}[x]$, а значит, операция замены переменных является корректной операцией в кольце полиномов.

Теорема 4. Операция замены переменных порождает гомоморфизм кольца $\mathbb{K}$ в себя.

Доказательство. Пусть $a(x), b(x) \in \mathbb{K}[x]$ – полиномы степени m, $p(x) \in \mathbb{K}[x]$ – полином степени k. Пусть P – отображение, порождаемое заменой переменных $p(x)$. Проверим, что $(a + b) \circ p = a \circ p + b \circ p$.

$$a \circ p + b \circ p = a\big(p(x)\big) + b\big(p(x)\big) =$$
$$= (a_m p(x)^m + \cdots + a_1 p(x) + a_0) + (b_m p(x)^m + \cdots + b_1 p(x) + b_0) =$$
$$= (a_m + b_m)p(x)^m + \cdots + (a_1 + b_1)p(x) + a_0 + b_0 = (a + b) \circ p$$

Свойство $(a \times b) \circ p = a \circ p \times b \circ p$ доказывается аналогичным образом.

Также очевидно, что замена переменных переводит 1 в 1 и 0 в 0, что соответствует свойствам 3 и 4. Следовательно, P – гомоморфное отображение. ■

Утверждение 2. Композиция гомоморфизмов является гомоморфизмом.

Доказательство. Пусть f, g – гомоморфные отображения. Тогда их композиция $f\big(g(x \circ y)\big) = f\big(g(x) \circ g(y)\big) = f(g(x)) \circ f(g(y))$ также является гомоморфным отображением. ■

Таким образом, пользуясь, изложенным теоритическим материалом построим схему 2 гомоморфного шифрования.

2.4. Схема 2

Генерация ключей. Пусть x_0 – закрытый ключ данного шифра, $x_0 \in \mathbb{A}$, алгебраическое случайное число.

Шифрование. Рассмотрим алгоритм шифрования $Enc : \mathbb{Z} \to \mathbb{Z}[x]$.

1) Пусть $z \in \mathbb{Z}$ – число, которое требуется зашифровать.
2) Выберем неприводимый над полем $\mathbb{Z}$ полином $u(x)$, такой, что $u(x_0) = 0$. Это можно сделать следующим образом. К примеру, если $x_0 = \sqrt{2} + \sqrt{3}$, то полином $u(x)$ строится следующим образом:

$$x_0{}^2 = 2 + 3 + 2\sqrt{6} = 5 + 2\sqrt{6}$$

$$x_0{}^2 - 5 = 2\sqrt{6}$$
$$(x_0{}^2 - 5)^2 = 24$$
$$x_0{}^4 - 10x_0{}^2 + 25 = 24$$
$$x_0{}^4 - 10x_0{}^2 + 1 = 0$$
$$u(x) = x^4 - 10x^2 + 1$$

3) Сделаем замену переменных $x = y - c$, где c – произвольное число.
4) Построим шифрующий полином $g_z(x)$ как отображение числа z согласно алгоритму 3 с использованием полинома $u'(x) = u(x + c)$, получившегося после замены переменной.

Расшифровка. Рассмотрим алгоритм дешифровки $Dec: \mathbb{Z}[x] \to \mathbb{Z}$.

4) Пусть полином $g''_{z'}(x) \in \mathbb{Z}[x]$ – зашифрованные данные, $x_0 \in \mathbb{A}$ – закрытый ключ, с $\in \mathbb{A}$ - известное число, использовавшееся во время замены переменной.
5) Для расшифровки необходимо вычислить $g''_{z'}(x)$ в точке $x_0 - $ с.
6) Тогда $g''_{z'}(x_0 - \text{с}) = s(x_0 - c)u'(x_0 - c) + z' = s(x_0 - c)u(x_0 - c + c) + z' = s(x_0 - c)u(x_0) + z' = z' \in \mathbb{Z}$ – расшифрованные данные.

Согласно рассмотренным в пункте 2.3. утверждениям данное шифрование также является гомоморфным.

Данную схему можно рассматривать как обобщение схемы 1.

2.5. Реализация схем в приложении

В приложении «Защищенный калькулятор» данные схемы применяются следующим образом.

Пусть клиенту требуется вычислить функцию $f(x_1, \dots x_m) \in \mathbb{Q}$, содержащую операции сложения, умножения, вычитания, деления в точке $z = (z_1, \dots, z_m)$, где $z_i \in \mathbb{Z}$.

1) Согласно каждой из схем выбирается закрытый ключ x_0.
2) Согласно каждой из схем выбираются m шифрующих полиномов $g_1, \dots g_m$:

В случае схемы 1:

$g_1(x_0) = a_{00} + a_{01}x_0 + a_{02}x_0^2 + \cdots + a_{0n}x_0^n = z_1$

$g_2(x_0) = a_{10} + a_{11}x_0 + a_{12}x_0^2 + \cdots + a_{1n}x_0^n = z_2$

…

$g_m(x_0) = a_{m-1\,0} + a_{m-1\,1}x_0 + a_{m-1\,2}x_0^2 + \cdots + a_{m-1\,n}x_0^n = z_m$

В случае схемы 2:

$g_1(x) = s_1(x)u(x) + z_1$

$g_2(x) = s_2(x)u(x) + z_2$

...

$g_m(x) = s_m(x)u(x) + z_m$

3) Представление функции $f(x_1, \dots x_m)$ и набор полиномов $g_1(x), \dots g_m(x)$ отправляются на сервер.
4) На сервере происходит подстановка полиномов $g_1(x), \dots g_m(x)$ в функцию $f(x_1, \dots x_m)$ и вычисляется функция $f\big(g_1(x), \dots g_m(x)\big) = Res(x)$.
5) $Res(x)$ отправляется клиенту.
6) Клиент вычисляет значение $Res(x)$ в секретной точке x_0 (для схемы 1), либо $x_0 - c$ (для схемы 2) и получает искомый результат $f(z_1, \dots z_m)$, согласно процедурам дешифровки для каждой из схем.

2.6. Анализ криптографической стойкости схем

Данные схемы требуется проанализировать на криптографическую стойкость по отношению к следующим видам атак:

- атака на основе только шифротекста (полный перебор);
- атака с подобранным открытым текстом;
- атака с подобранным зашифрованным текстом.

Рассмотрим перевод определений, предложенных в [4].

Определение 6. ***Атака на основе только шифротекста.*** При выполнении этой атаки криптоаналитику доступно некоторое количество шифротекстов, являющихся результатом применения одного алгоритма шифрования. Задача криптоаналитика заключается в том, чтобы найти как можно больше открытых текстов, соответствующих имеющимся шифротекстам, а в лучшем случае – определить ключ, использованный при шифровании.

Допустим, криптоаналитик обладает некоторым числом шифротекстов. Основной защитой от атаки полным перебором является тот факт, что происходит шифрование целого числа, а не текстовых данных, поэтому со стороны потенциального взломщика возникает сложность в вопросе определения правильности того или иного ответа.

В схеме 1 в случае целого закрытого ключа для проведения атаки полным перебором требуется перебрать все числа $x_0 \in \mathbb{Z}$. Для случая $x_0 = \frac{p}{q}$, $x_0 \in \mathbb{Q}$ задача усложняется, так как требуется перебрать все возможные пары чисел (p, q). Количество чисел в машинном изображении множества целых чисел зависит от реализации используемых библиотек длинной арифметики, то есть, в данном случае ограничено объемом оперативной памяти.

Для схемы 2 закрытый ключ $x_0 \in \mathbb{A}$ является алгебраическим числом. В данном случае имеем аналогичную ситуацию: представление алгебраических чисел в используемых библиотеках также ограничено размером оперативной памяти.

Таким образом, алгоритмы 1,2 можно считать устойчивыми по отношению данному виду атак.

Определение 7. ***Атака с использованием выбранного открытого текста.*** У криптоаналитика есть доступ к шифротекстам и открытым текстам нескольких сообщений. Его задача состоит в получении ключа (или ключей), использованного для шифрования сообщений, для дешифрования других сообщений, зашифрованных тем же ключом (ключами).

Определение 8. ***Атака с использованием выбранного шифротекста.*** Криптоаналитик может выбрать различные шифротексты для дешифрования и имеет доступ к дешифрованным открытым текстам. Его задача состоит в получении ключа.

К сожалению, данные виды атак могут быть успешно применены к криптографической схеме 1.

При наличии у злоумышленника открытого текста и шифротекста одновременно возможно решить уравнение $g_z(x) = q^n f(x) - q^n f\left(\frac{p}{q}\right) + z = z$, и найти ключ.

Однако смена ключа происходит после каждой сессии, что исключает возможность использовать выявленный в результате атаке ключ в целях злоумышленника.

В криптографической схеме 2 решение уравнения $g_z(x) = s(x)u(x) + z = z$ осложняется тем, что оно имеет алгебраические корни. Кроме того, данные атаки можно предотвратить путем сокрытия полинома замены. В рассматриваемой схеме используется полином степени $l = 1$. При степени полинома замены равной l, потребуется перебрать все возможные кортежи целых чисел размерности l, что потребует еще больших временных затрат, чем для атаки только на основе шифротекста. Данный подход увеличит криптографическую стойкость схемы.

2.7. Оценка сложности операций в криптосистеме

Произведем оценку алгоритмической сложности выполнения операций $+$ и $\times$ над зашифрованными данными. Пусть для двух чисел a_0, $b_0 \in \mathbb{Z}$ – числа, которые требуется зашифровать. Пусть соответствующие им шифрующие полиномы $a(x)$, $b(x) \in \mathbb{Z}[x]$ имеют степени n и m.

Тогда для схемы 1 операция сложения будет иметь алгоритмическую сложность $O(n+m)$, операция умножения – $O(nm)$.

Для схемы 2 пусть a_0, $b_0 \in \mathbb{Z}$ – числа, которые требуется зашифровать $a(x)$, $b(x) \in \mathbb{Z}[x]$ соответствующие им шифрующие полиномы степени n и m, $p(x)$ – полином замены степени $l > 0$. Тогда операция сложения будет иметь сложность $O(l(n+m))$, умножения – $O(nml^2)$, в случае $l = 1$ сложность операций – $O(n+m)$ и $O(nm)$ соответственно.

Заметим, что при сложении двух полиномов степень результата не превышает степень слагаемых, а при умножении она равна сумме степеней множителей. Из этого можно сделать вывод, что количество выполненных сложений не оказывает негативного влияния на производительность вычислений, а умножение приводит к замедлению вычислений. Для уменьшения скорости роста степеней в качестве начальных шифрующих полиномов предлагается генерировать полиномы минимальной степени. То есть, в первом случае предлагается использовать линейные полиномы, во втором – полином $u(x)$ должен быть квадратичным, полином $s(x)$ и полином замены должны быть линейными. В силу рассмотренной в пункте 2.6 теории это возможно сделать без ущерба для безопасности.

Глава 3. Разработка экспериментальной реализации

3.1. Приложение «Защищенный калькулятор»

Реализация предложенных алгоритмов является важной частью работы, так как является единственной возможностью на практике оценить их эффективность. В разделе 1.4. уже были описаны технические требования к клиентской компоненте приложения. В этом разделе будет освещена архитектура приложения и различные технические решения, которые предъявлялись к реализации.

Приложение «Защищенный калькулятор» состоит из клиентского и серверного компонентов, а также библиотеки полиномов.

Клиентский компонент

Клиентская часть представляет приложение с графическим пользовательским интерфейсом, представляющим калькулятор. Оно предоставляет пользователю возможность ввести математическое выражение, состоящее из операций +, ×, –, ÷ и целых чисел, а также скобок. В клиентской части происходит генерация секретных ключей и шифрующих полиномов согласно приведенным схемам. Далее набор полиномов и представление функции передается на сервер.

Получив ответ от сервера, клиентский компонент производит операцию расшифровки и предоставляет результат клиенту.

Библиотека работы с полиномами

Обе части приложения используют библиотеку математических операций над полиномами. Она включает реализацию полиномов, полиномиальных дробей и арифметических операций над ними, в том числе: операции сложения, умножения, вычитания, деления. Коэффициенты полиномом представляются числами из длинной арифметики.

Операции сложения и умножения были реализованы с помощью классических алгоритмов, работающих за время $O(n)$ и $O(n^2)$ соответственно (операция вычитания полностью аналогична операции сложения, а операция деления – операции умножения). Операция умножения на константу выполняется за время $O(n)$. Для вычисления значения полинома в точке используется схема Горнера [5], позволяющая минимизировать число операций умножения и вычисляющая искомое значение за $O(n)$.

Серверный компонент

Основу данной работы составляет разработка серверного компонента приложения. В серверной части происходит параллельная обработка клиентских запросов, вычисление функции от зашифрованных данных и отправка результата вычислений клиенту.

Рассмотрим архитектуру серверного компонента более подробно.

3.2. Архитектура серверной части приложения

Серверная часть приложения состоит из четырех основных пакетов.

ru.nsu.parallels.protectedCalculator.connectionManager – пакет содержащий, набор классов управляющий соединениями клиентов.

Протокол передачи данных

Для передачи информации от клиентского компонента к серверному был создан протокол прикладного уровня HEP *(Homomorphic Encryption Protocol),* использующий транспортный протокол TCP. В качестве основного протокола был выбран TCP, так как он позволяет обеспечить эффективную и надежную доставку по длинному каналу передачи данных. В реализации схемы взаимодействия клиента и сервера используется стандартный механизм сокетов языка Java. Порт, используемый приложением, может задаваться пользователем. По умолчанию предлагается использовать незарезервированный порт 6060.

Основой прикладного протокола является технология «клиент-сервер», то есть предполагается существование потребителей (клиентов), которые инициируют соединение и посылают запрос, и поставщика (сервера), который ожидает соединения для получения запроса, производит необходимые действия и возвращает обратно сообщение с результатом.

Рассмотрим структуру прикладного протокола. Каждое сообщение клиента состоит из следующих частей:

1) длина данных;
2) набор шифрующих полиномов;
3) представление функции, задаваемой пользователем;

В ответ сервера входит:

1) результат выполнения операции (в случае неуспешного выполнения – код ошибки);
2) длина данных (в случае успешного выполнения операции);
3) результат вычислений (в случае успешного выполнения операции).

Управление соединениями

Одной из упрощенных моделей для построения серверных приложений является создание нового потока каждый раз, когда появляется запрос клиента и обслуживание запроса в новом потоке. Этот подход в действительности хорош для разработки прототипа приложения, но имеет значительные недостатки. К основным недостаткам такой модели можно отнести:

- значительные системные издержки частого создания и завершения потоков;
- малое время работы потока;
- нерегулируемое количество потоков;
- в большинстве случаев отсутствие очереди клиентских запросов;
- большое количество переключений контекстов рабочих потоков.

Для решения этих проблем предназначен пул потоков.

Рассмотрим механизм работы пула потоков. Имеется главный поток приложения, прослушивающий клиентские запросы. Пул потоков создается заранее или при поступлении первого запроса. При поступлении запроса главный поток выбирает поток из пула и передает ему запрос. Если количество потоков в пуле достигло максимума, запрос помещается в очередь. Если количество потоков меньше максимального, и все они заняты обработкой, создается новый поток, который получает клиентский пакет на обработку. Если количество потоков равно максимальному и все потоки активны, пакет добавляется в очередь и ожидает освобождения одного из потоков. Алгоритмы добавления потоков в пул и определения оптимального размера пула зависят от решаемой задачи.

Преимущество этого подхода в том, что пул потоков сам управляет необходимым количеством потоков, обрабатывает их пуск и завершение работы, а также проблемы, связанные с надежностью.

В серверном компоненте «Защищенный калькулятор» для управления клиентскими соединениями был реализован фиксированный пул потоков посредством открытой библиотеки утилит *java.util.concurrent*[1]. Класс *ExecutorService* из этого пакета – эффективная, широко использующаяся, правильная реализация пула потоков, основанного на рабочей очереди.

[1]Официальный сайт: http://docs.oracle.com/javase/6/docs/api/java/util/concurrent/package-summary.html

ru.nsu.parallels.protectedCalculator.gui – пакет, содержащий набор классов, реализующих графический интерфейс данного приложения. Пользователю предоставляется информация об активных соединениях, времени произведения операций, наглядно отображаются текущие вычисления над зашифрованными данными, представленными в виде полиномов. Для реализации графического интерфейса была использована библиотека Java Swing.

ru.nsu.parallels.protectedCalculator.calculations – пакет, содержащий классы для выполнения вычислений над зашифрованными данными. Использует библиотеку для произведения математических операций над полиномами *Polynomial Math Library*.

Основной составляющей данного пакета является класс ***FunctionalParser***, который производит анализ представления функции и вычисляет значение для заданного набора полиномов. Для анализа математического выражения была использована open-source библиотека Java.Apache Commons JEXL[1] (Java Expression Language). JEXL – это библиотека, целью которой является упрощение реализации «скриптовых решений» в приложениях и фреймворках, написанных на языке Java. JEXL реализует язык выражений (Expression Language), основанный на расширении JSTL Expression Language. Библиотека содержит мощные средства для синтаксического анализа математических выражений и имеет подходящий для данной задачи API. JEXL предоставляет возможность определения собственных переменных любого типа, функций, а также изменение поведения существующих операторов.

Также была создана собственная экспериментальная реализация синтаксического анализатора представления функции, задаваемой пользователем, использующая метод рекурсивного спуска. Однако анализатор, использующий JEXL, превзошел по производительности данную реализацию, в результате чего было принято решение включить в конечный продукт синтаксический анализатор на основе JEXL.

ru.nsu.parallels.protectedCalculator.logger – пакет классов, отвечающий за запись информации о проводимых вычислениях в лог. К информации, которая подлежит занесению в текстовый лог-файл, относится IP-адрес клиентского соединения, время вычисление заданной пользователем функции, представление функции, шифрующие полиномы. В результате пользователю приложения предоставляется возможность проводить анализ большого количества произведенных вычислений.

[1]Официальный сайт JEXL: http://commons.apache.org/proper/commons-jexl/

3.3. Криптографическая библиотека на языке C++

Для тестирования производительности было решено создать аналог криптографических примитивов в виде библиотеки на языке C++ с использованием длинной арифметики библиотеки GMP[1].

Язык C++ в отличие от Java является языком со статической компиляцией, вследствие чего обладает большей производительностью. Приложения Java обладают хорошей переносимостью и достаточно высокой скоростью работы. Однако даже при наличии JIT-компиляции в алгоритмических задачах они работают медленнее, чем программы, написанные на C или C++. Это связано с тем, что JIT-компиляция происходит «на лету» в условиях жесткой ограниченности времени и ресурсов и создает не такой оптимальный код как многопроходный компилятор C/C++, который может тратить достаточно большое время и количество ресурсов на отыскание конструкций программы, которые можно оптимизировать. Кроме того, JIT-компиляция увеличивает потребления памяти для хранения результатов компиляции.

К преимуществам языка C++ также можно отнести:

1) Возможность создания обобщённых алгоритмов для разных типов данных, их специализация и вычисления на этапе компиляции, используя шаблоны.
2) Большое количество компиляторов для различных платформ.
3) Эффективность. Язык спроектирован так, чтобы дать программисту максимальный контроль над всеми аспектами структуры и порядка исполнения программы. Ни одна из языковых возможностей, приводящая к дополнительным накладным расходам, не является обязательной для использования — при необходимости язык позволяет обеспечить максимальную эффективность программы.

Библиотека GMP является свободной библиотекой арифметики произвольной точности, включающей операции над целыми знаковыми, рациональными и действительными числами. Ограничением длины чисел является объем оперативной памяти устройства, на котором исполняется библиотека GMP. Также GMP имеет богатый набор различных математических функций.

Основными целями создания приложений, использующих библиотеку GMP, являются: создание криптографических и исследовательских приложений,

[1]Официальный сайт GMP: http://gmplib.org/

Интернет-приложений безопасности, алгебраических систем, исследование вычислительных методов алгебры и т. д.

GMP обеспечивает высокую производительность операций над числами любых размеров. Высокая скорость операций достигается за счет использования полного машинного слова в качестве основного арифметического типа, ускоренных с помощью ассемблерных вставок алгоритмов, реализованных под различные архитектуры процессоров, и других оптимизаций, направленных на увеличение скорости вычислений.

Поддерживаемые платформы: 32-разрядной и 64-разрядной версии Windows, Unix-подобные операционные системы, такие как GNU/Linux, Solaris, HP-UX, Mac OS X/Darwin, BSD, AIX, и т. д.

В данной работе было решено использовать версию MPIR[1] библиотеки GMP, разработанную для компилятора Microsoft Visual C++.

Криптографическая библиотека включает следующие основные модули:

1) Шифрования, дешифрования. Класс Encryptor предоставляет методы для шифрования и дешифрования данных в соответствии с каждой из схем, рассмотренных в главе 2.
2) Операции с полиномами. Класс Polynomial определяет представление полинома и реализует операции сложения, умножения, вычитания, деления над полиномами.
3) Операции с полиномиальными дробями. Класс PolynomialFraction определяет представление полиномиальной дроби и реализует операции сложения, умножения, вычитания, деления над полиномиальными дробями.

Для хранения коэффициентов полинома используется динамически выделяемая память, для управления которой используется класс std::vector из стандартной библиотеки языка C++.

Для вычислений над зашифрованными данными были реализованы операции сложения, умножения, вычитания и деления полиномов и полиномиальных дробей. Все арифметические операторы были реализованы в двух вариантах: с размещением результата в новом объекте (+, -, *, /) и с присваиванием результата в левый операнд (+=, -=, *=, /=). Использование последнего вида операторов позволяет уменьшить количество выделяемой памяти и тем самым увеличивает производительность вычислений. Передача параметров в операторы происходит по ссылке, что позволяет не тратить

[1]Официальный сайт MPIR: http://mpir.org/

процессорное время и память на создание лишних копий объектов.

Для создания шифрующих полиномов и генерации секретного ключа используется генератор (псевдо)случайных чисел. Генератор псевдослучайных чисел из стандартной библиотеки признан криптографически ненадежным, так как использует линейно конгруэнтный метод. Поэтому был введен интерфейс, позволяющий использовать произвольные реализации генератора. По умолчанию рекомендуется использовать генератор случайных чисел из библиотеки Boost[1]. Он использует недетерминированные источники случайных чисел и является криптографически надежным.

Для проверки правильности написанных алгоритмов над полиномами и полиномиальными дробями библиотека была протестирована с помощью модульного тестирования. В настоящее время для тестирования кода на языке C++ существует большое количество инструментов и библиотек, наиболее популярными из них являются:

- Google Test;
- CppUnit;
- CppTest;
- CxxTest.

Для тестирования разрабатываемой библиотеки была выбрана библиотека Google Test[2], так как она является довольно простой в использовании, обладает лаконичным синтаксисом создания тестов и обладает функциональностью, достаточной для данного проекта.

Для обеспечения высокой эффективности модульного тестирования, для каждого класса библиотеки был разработан набор тестов, покрывающий все его публичные методы. Такой подход позволил обеспечить 100% покрытие строк кода и обеспечить эффективное обнаружение ошибок в процессе рефакторинга и оптимизации производительности.

3.4. Анализ производительности

Оценка производительности в данном исследовании схем гомоморфного шифрования является важной задачей данного исследования.

В работе использовалось два вида тестов производительности:

[1]Официальный сайт Boost: http://www.boost.org/

[2]Официальный сайт Google Test: https://code.google.com/p/googletest/

1) Тесты, измеряющие время выполнения одной и той же операции для зашифрованных и незашифрованных данных.
2) Деградационные тесты – тесты, измеряющие зависимость времени выполнения одной операции в серии от числа операций в этой серии.

Тестирование производительности производилось под управлением ОС Windows 7 на компьютере с двухъядерным процессором Mobile DualCore AMD Phenom II N620 с частотой каждого ядра 2793 МГц и объемом RAM 4 Гб. Для компиляции криптографической библиотеки использовался компилятор MS Visual C++ 2010.

За основу сценариев тестирования были выбраны операции сложения и умножения над линейными полиномами.

Всего рассматривались 4 вида различных тестов:

- Два теста на сравнение производительности операции *сложения* и *умножения* над зашифрованными данными и незашифрованными данными; вычислялось среднее время операции за 10^8 итераций.
- Два теста на исследование зависимости скорости операций *сложения* и *умножения* от числа выполненных операций.

В первом наборе тестов рассматривали ключи размером 2^n, где $n = 64; 128; 256; 512; 1024; 2048; 4096$ и четыре операции: сложение и умножение с созданием нового объекта $(+, -)$, а также аналогичные операции, но с записью результата в левый операнд $(+=, *=)$. Результаты первого набора тестов представлены в таблице 1.

В таблице отражено отношение среднего времени одной операции над шифрующими линейными полиномами к среднему времени одной операции над незашифрованными данными. Рассматривалась различная длина ключей и, как следствие, коэффициентов полиномов.

Время операций в криптосистеме Таблица 1.

Операции/ Длина ключа 2^n	**64**	**128**	**256**	**512**	**1024**	**2048**	**4096**
$+=$	350	500	650	700	800	1000	1200
$+$	2000	2800	3000	4000	4500	4800	5000
$*=$	800	1000	3000	5800	16000	50000	100000
$*$	3000	6000	7500	12000	30000	80000	200000

Данное падение производительности является ожидаемым и приемлемым на этапе экспериментального исследования. Низкая производительность операции умножения по сравнению с операцией сложения обусловлена более высокой сложностью алгоритма. Также было выявлено, что операции с памятью занимают большую часть процессорного времени. Одним из возможных решений в дальнейшем может быть создание собственного аллокатора, оптимизированного под данную задачу.

Отсюда можно сделать вывод, что при дальнейшей оптимизации, данные схемы могут быть пригодны для применения в реальных программных продуктах.

Второй набор тестов отражает зависимость скорости вычислений от количества выполненных операций. Были использованы тесты, параметризованные количеством операций, выполняемых в одной цепочке. Начальный размер ключа и коэффициентов шифрующих полиномов составляет 2^{64}.

На рисунке 1 изображен график зависимости времени, приходящегося на одну операцию сложения, от количества операций в серии. На графике виден линейный рост, который является следствием увеличения объема памяти, выделяемой под коэффициенты полинома в результате сложения, а также усложнением операции сложения над данными большей длины. Однако так как операция сложения полиномов имеет алгоритмическую сложность $O(n)$, данный рост можно считать несущественным. Так, на миллион итераций наблюдается падение скорости на 0.7 с, что не приводит к заметному снижению производительности.

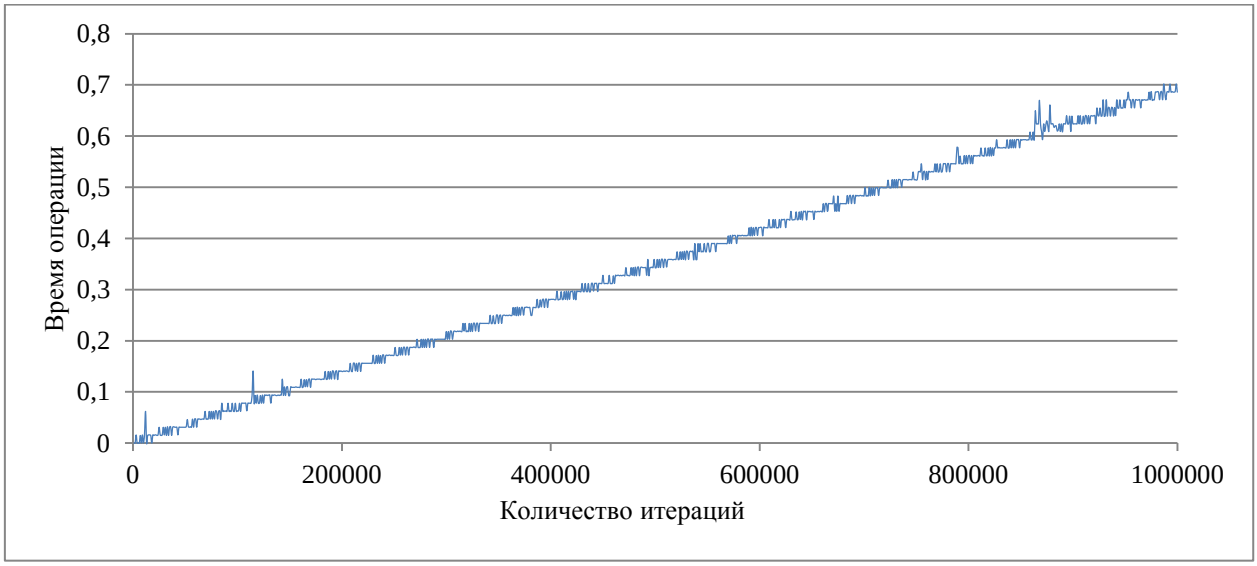

Рисунок 1. Зависимость времени выполнения операции сложения от длины цепочки.

На рисунке 2 изображен график аналогичный график для операции умножения. Здесь виден квадратичный рост и, в отличие от первого случая, наблюдается более значительное снижение скорости. Это объясняется тем, при умножении полиномов умножаются их коэффициенты, что требует выделения большего объема памяти, чем для сложения коэффициентов, а также операция умножения полиномов имеет более высокую алгоритмическую сложность – $O(n^2)$.

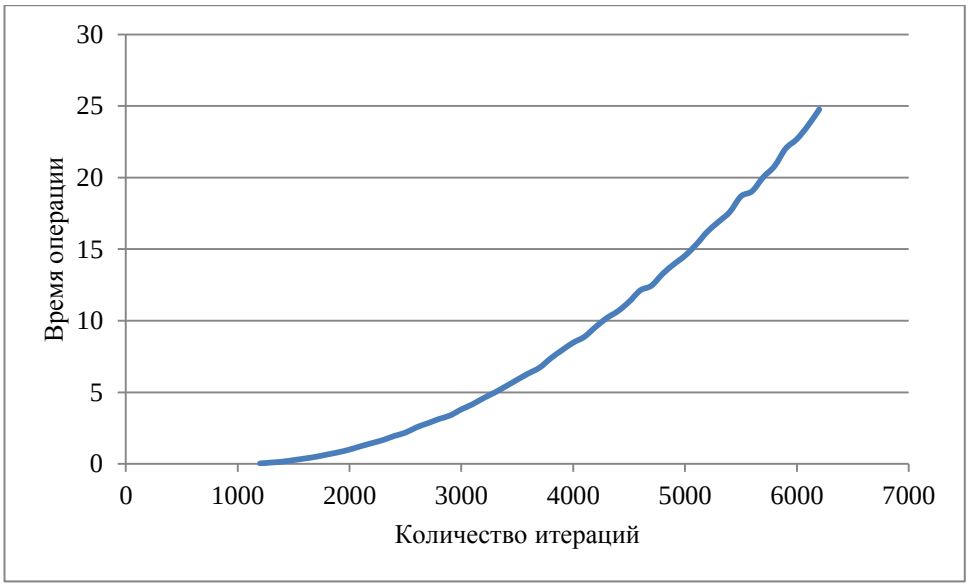

Рисунок 2. Зависимость времени выполнения операции умножения от длины цепочки.

Глава 4. Заключение

В результате исследования были получены следующие результаты:

- исследована существующая схема гомоморфного шифрования, предложенная Гентри;
- разработаны и теоретически обоснованы две схемы полностью гомоморфного шифрования;
- исследована криптографическая стойкость данных схем к различным видам атак;
- реализован серверный компонент приложения «Защищенный калькулятор», производящий операции над зашифрованными данными;
- проанализирована производительность экспериментальной реализации схем;
- выявлены проблемы схем, которые предлагается исследовать в дальнейшем.

По результатам исследования можно сделать вывод, что данные схемы представляют перспективную разработку и являются одним из возможных вариантов применения полностью гомоморфного шифрования на практике. Тем не менее, они обладают недостатками, присущими любым схемам полностью гомоморфного шифрования, основанных на полиномах. Устранение этих недостатков является приоритетным направлением для дальнейших исследований.

Таким образом, в будущем планируется:

- исследовать проблему роста степеней и возможные решения (например, с помощью фактор-кольца многочленов);
- улучшить устойчивость схемы к различным видам криптоанализа;
- использовать графические ускорители для повышения производительности вычислений;
- использовать полученные результаты, для создания защищенной СУБД.

Литература

1. Mell, Peter. The NIST Definition of Cloud Computering / Peter Mell, Timothy Grance. – 2011. – Sept. URL: http://csrc.nist.gov/publications/nistpubs/800145/SP800145.pdf
2. Rivest, R.L. On data banks and privacy homomorphisms / R.L. Rivest, L. Adleman, M.L. Dertouzos // Foundations of secure computation.— 1978.— Vol. 32, no. 4.— Pp. 169–178. URL: http://people.csail.mit.edu/rivest/pubs/RAD78.pdf.
3. Craig Gentry. A fully homomorphic encryption scheme // PhD thesis. – Stanford University. – 2009. – URL: http://crypto.stanford.edu/craig
4. Шнайдер, Б.А. Прикладная криптография. – М.: Триумф, 2002. -816 с.
5. Ананий В. Левитин Глава 6. Метод преобразования: Схема Горнера и возведение в степень // Алгоритмы: введение в разработку и анализ = Introduction to the Design and Analysis of Algorithms. – М.: «Вильямс», 2006. с. 284-291.
6. Скляров, Д. Искусство защиты и взлома информации. – СПб: БХВ-Петербург, 2004, -271 с.
7. Варновский Н.П., Шокуров А.В. Гомоморфное шифрование. //Труды Института Системного программирования: Том 12. (под Ред. В.П. Иванникова) – М.: ИСП РАН, 2006, с. 27-36.