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

Яковлев Михаил Олегович

Защищенный калькулятор. Разработка клиентского компонента

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

Руководитель	**Автор**
Кренделев С.Ф.	Яковлев М.О.
……..	……..
(фамилия, И.О.)	(фамилия, И.О.)
к. ф.-м. н., доцент	ФИТ, 9201
……..	……..
(уч.степень, уч.звание)	(факультет, группа)
……..	……..
(подпись, дата)	(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

Студенту Яковлеву Михаилу Олеговичу

Направление подготовки 230100.62 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА

ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ

Тема: «Защищенный калькулятор. Разработка клиентского компонента»

Исходные данные (или цель работы): разработка и исследование свойств схемы полностью гомоморфного шифрования на основе полиномов в кольце целых чисел, реализация данной схемы в виде приложения «Защищенный калькулятор».

Структурные части работы: разработка математической модели данной схемы в виде алгоритмов шифрования, дешифрования, анализ криптоустойчивости данных алгоритмов к различным видам атак, реализация данной схемы в виде клиентского компонента приложения «Защищенный калькулятор», анализ производительности.

Содержание

ВВЕДЕНИЕ

Одним из наиболее перспективных направлений развития информационных технологий на текущий год являются «облачные вычисления», т.е. практика аренды вычислительных мощностей у сторонних провайдеров, которые берут на себя всё обеспечение работы инфраструктуры, резервирование и т.п.

Данный подход экономически выгоден, поскольку освобождает потребителя облачных вычислений от необходимости содержать собственные сервера, обеспечивать высокий уровень их работоспособности, заниматься вопросами электроэнергии, охлаждения, пожарной безопасности и многими другими. Провайдер облачного сервиса, в свою очередь, обеспечивает обслуживание серверов на высоком уровне качества, который недоступен небольшим организациям. Кроме того, за счет автоматизации модификации выделения ресурсов организация-потребитель получает возможность гибко реагировать на изменения вычислительных способностей.

Ниже приведен частичный перевод определения облачных вычислений, предложенного NIST[1].

Определение 1. Облачные вычисления – это модель организации полноценного, удобного и предоставляемого по необходимости (on-demand) доступа по сети к разделяемому пулу настраиваемых вычислительных ресурсов (сетям, серверам, хранилищам данных, приложениям и сервисам), которые могут быть быстро получены и освобождены с минимальными усилиями (management effort) или взаимодействием с провайдером. Необходимыми условиями облачных вычислений являются: on-demand self-service, broad network access, resource pooling, rapid elasticity, measured service.

Тем не менее, концепция облачных вычислений подвергалась критике. Из-за того, что обслуживанием серверов занимается третья сторона, появляется целый класс новых проблем, связанных с информационной безопасностью. Наиболее серьезной из них является проблема конфиденциальности данных. Поскольку парк серверов, на которых работает облачный сервис, находится на территории провайдера, организация-потребитель, которая размещает свои данные в облаке, не может контролировать физический доступ к ним. Ситуация осложняется тем, что законодательством многих стран предписываются жесткие требования к уровню защиты данных определенного класса информации, которые трудно или невозможно обеспечить в случае, когда данные хранятся в облаке.

Единственным решением проблемы конфиденциальности данных, доступным на данный момент, является шифрование данных, находящихся в облаке. Однако криптографические системы, доступные в настоящее время, не позволяют полноценно использовать зашифрованные данные. В результате, их приходится расшифровывать, что оставляет только следующие сценарии использования облаков:

- Облако хранит зашифрованную информацию. Для доступа к ней данные скачиваются на доверенный компьютер, после чего расшифровываются и обрабатываются.

  Подобный подход может быть оправдан только в случае, когда необходимо хранить большой объем редко используемой информации, так как для любых манипуляций с ней необходимо их скачать через глобальную сеть, а затем расшифровывать, что значительно ниже скорости доступа к локальному хранилищу.
- Облако также хранит зашифрованную информацию, но для её обработки часть данных расшифровывается, а затем зашифровывается прямо на облаке.

  У такого подхода сразу два недостатка: ключ, которым происходит шифрование, попадает в облако и может быть перехвачен злоумышленником; расшифрованные данные во время обработки находятся в облаке в открытом виде и так же могут быть перехвачены. Таким образом, весь смысл шифрования теряется, особенно в случае частого обращения к данным.

Возможным решением проблемы конфиденциальности данных может стать алгоритм полностью гомоморфного шифрования. Особенностью этого вида шифрования является возможность производить над зашифрованными данными основных математических операций – сложения и умножения – таким образом, что после расшифровки результат будет соответствовать результату, который был бы получен, если бы действия проводились над нерасшифрованными данными. Более формальное определение полностью гомоморфного шифрования будет приведено далее.

На данный момент существует несколько частично гомоморфных систем шифрования, т.е. алгоритмов, удовлетворяющих части требований, предъявляемых полностью гомоморфному шифрованию. Например, такие криптосистемы, как RSA и криптосистема Эль-Гамаля обладают свойством гомоморфности для операции умножения. Криптосистема Крейга Гентри обладает гомоморфностью для операции

умножения, а также позволяет производить над данными ограниченное число операций сложения.

Кроме того, в 2009 году Крейг Гентри предложил полностью гомоморфный алгоритм шифрования, который, тем не менее, очень требователен к ресурсам процессора и памяти, а потому, не применим на практике. На самом деле схема Гентри не является в математическом смысле гомоморфным шифрованием. Правильнее было бы назвать его схему, схемой секретных вычислений.

Таким образом, проблема создания полностью гомоморфной криптографической системы по-прежнему является открытой и актуальной.

В рамках данной дипломной работы были поставлены следующие задачи: разработка и совершенствование (повышение криптоустойчивости и быстродействия) алгоритмов полностью гомоморфного шифрования, реализация алгоритмов в виде клиент-серверного приложения «Защищенный калькулятор», исследование криптостойкости и быстродействия разработанных алгоритмов.

Глава 1. Постановка задачи

1.1. Математическая формулировка задачи

В основе понятия «гомоморфное шифрование» находится понятие «privacy homomorphism», введенное в [2]:

Определение 2 (Privacy homomorphism). Пусть даны две алгебраические системы:

$$U = \langle S; f_1, \dots, f_k;\ p_1, \dots, p_l;\ s_1, \dots, s_m \rangle$$

$$C = \langle S'; f'_1, \dots, f'_k;\ p'_1, \dots, p'_l;\ s'_1, \dots, s'_m \rangle$$

где S и S' – множества; $f_1, \dots, f_k$, $f'_1, \dots, f'_k$ - функции; $p_1, \dots, p_l, p'_1, \dots, p'_l$ - предикаты, определённые на множествах S и S' соответственно, а $\{s_1, \dots, s_m\} \subset S$, $\{s'_1, \dots, s'_m\} \subset S'$ - известные константы.

Пусть существует обратимая функция $\varphi: S \to S'$. Тогда φ называется privacy homomorphism, если выполняются следующие условия:

$$\forall i \in \{1, \dots, k\} : f_i(a_1, \dots, a_n) = \varphi^{-1}\left(f'_i\left(\varphi(a_1), \dots, \varphi(a_n)\right)\right)$$

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

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

а также:

а) функции φ и φ^{-1} легко вычислимы;

б) операции f'_i и предикаты p'_i так же эффективно вычислимы;

в) представление «зашифрованного» значения $\varphi(d_i)$, где $d_i \in S$, занимает не слишком много места по сравнению с d_i;

г) знания $\varphi(d_i)$ для большого количества d_i недостаточно для восстановления φ (атака с известным шифротекстом)

д) знание некоторых пар $\left(d_i, \varphi(d_i)\right)$ недостаточно для восстановления φ (атака с подобранным открытым текстом)

е) знания алгоритмов вычисления операций f'_i и предикатов p'_i недостаточно для построения алгоритма вычисления φ.

Определим понятие гомоморфного шифрования:

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

а) существует достаточно большое количество функций φ, удовлетворяющих определению privacy homomorphism, при чём такие функции достаточно легко строить;
б) число операций $l > 0$;

Иными словами, гомоморфное шифрование – это разновидность шифрования, которое позволяет выполнять как минимум одну математическую операцию таким образом, что после расшифровывания результат совпадет с результатом применения той же операции к незашифрованным данным.

Определение 4 (полностью гомоморфное шифрование). Полностью гомоморфное шифрование – это гомоморфное шифрование, где S и U – кольца, причём среди операций f_i существуют операции сложения и умножения.

1.2. Технические требования

В рамках дипломной работы требуется написать клиентскую компоненту приложения «Защищенный калькулятор», которая будет реализовывать указанные выше алгоритмы, и на её основе исследовать производительность полученной схемы шифрования.

Программа предназначена для вычисления заданных выражений, содержащих операции сложения, вычитания, умножения, деления, возведения в степень, в том числе со скобками путем отправки на Сервер зашифрованного исходного выражения и расшифровывания полученного в ответ результата.

В качестве языка реализации клиентской компоненты приложения был выбран язык Java. Причинами такого выбора являются:

- наличие стандартной библиотеки java.math и в частности классов BigDecimal и BigInteger, предоставляющих возможность использовать все преимущества длинной арифметики.

- богатые возможности для построения графического интерфейса (библиотека swing)
- кросс-платформенность

1.3. Ожидаемые результаты

В результате выполнения работы необходимо получить:

а) Общее описание алгоритмов шифрования
б) Предварительное заключение об устойчивости к различным атакам
в) Экспериментальную реализацию алгоритмов шифрования

Глава 2. Схемы полностью гомоморфного шифрования

В данном разделе разбирается схема Крэйга Гентри, как единственная на данный момент известная схема полностью гомоморфного шифрования. Затем рассматривается основная конструкция предлагаемых схем гомоморфного шифрования, анализируется их криптографическая устойчивость к различным видам атак, оценивается сложность производимых над зашифрованными данными вычислений.

2.1. Схема Крэйга Гентри

Принципиальную возможность полностью гомоморфного шифрования в 2009 году доказал исследователь из IBM Крэйг Гентри. В своей работе он построил схему, которую назвал *«Fully Homomorphic Encryption based on Ideal Lattices»*. Рассмотрим её на примере вычислений в Z_2, т.е. в вычислениях над битами.

В качестве секретного ключа выберем произвольное целое нечетное число p, оно будет иметь вид $p = 2k + 1$.

Пусть требуется зашифровать бит $m \in \{0,1\}$. Для этого построим число z по правилу $z = 2r + m$, где r – произвольно. Такое построение значит, что $z = m \; mod \; 2$. Шифрование заключается в том, что m сопоставляется число $c = pq + z$, где q - произвольно. Число c отправляется на вычисления.

Заметим, что $c \; mod \; 2 = (2r + m + (2k + 1)q) \; mod \; 2 = (2(r + kq) + m + q) \; mod \; 2 = (m + q) \; mod \; 2$. Поэтому возможно только узнать четность выхода из шифрования, но не более того.

Рассмотрим процесс расшифровки. Пусть есть число c, шифрующее неизвестный бит m. Предполагается, что p – секретный ключ – нам известен. Вычислим

$$c \; mod \; p = (z + pq) \; mod \; p = \; z \; mod \; p + (pq) \; mod \; p = z \; mod \; p = (2r + m) \; mod \; p$$
$$= 2(r \; mod \; p) + m \; mod \; p;$$

Число $c \bmod p$ называется «ошибкой», или «шумом». Нетрудно заметить, что его четность совпадает с исходным битом m:

$$(c \bmod p) \bmod 2 = (2(r \bmod p) + m \bmod p) \bmod 2 = (m \bmod p) \bmod 2 = m \bmod 2 = m;$$

Докажем, что данное шифрование является гомоморфным относительно операций сложения и умножения.

Пусть есть два бита $m_1, m_2 \in \{0,1\}$. Сопоставим им $z_1 = 2r_1 + m_1$, $z_2 = 2r_2 + m_2$. Выберем секретный ключ $p = 2k + 1$. За φ обозначим функцию шифрования.

Тогда $\varphi(m_1) = c_1 = z_1 + pq_1$, $\varphi(m_2) = c_2 = z_2 + pq_2$ – шифротексты для m_1 и m_2 соответственно. Их сумма и произведение будут выглядеть соответственно:

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

$$\varphi(m_1) \times \varphi(m_2) = c_1 c_2 = z_1 z_2 + p(z_1 q_2 + z_2 q_1) + p^2 q_1 q_2 = (2r_1 + m_1)(2r_2 + m_2) + p(z_1 q_2 + z_2 q_1) + p^2 q_1 q_2 = 4r_1 r_2 + 2(r_1 m_2 + r_2 m_1) + m_1 m_2 + p(z_1 q_2 + z_2 q_1) + p^2 q_1 q_2.$$

Заметим, что процедура их дешифровки дает соответственно сумму и произведение исходных бит m_1 и m_2:

$$\varphi^{-1}\big(\varphi(m_1) + \varphi(m_2)\big) = \big(2(r_1 + r_2) + m_1 + m_2 + p(q_1 + q_2)\big) \bmod p \bmod 2 = m_1 + m_2;$$

$$\varphi^{-1}\big(\varphi(m_1) \times \varphi(m_2)\big) = (4r_1 r_2 + 2(r_1 m_2 + r_2 m_1) + m_1 m_2 + p(z_1 q_2 + z_2 q_1) + p^2 q_1 q_2) \bmod p \bmod 2 = m_1 m_2$$

Тем не менее, следует заметить, что выполнение вычислений ведет к накоплению ошибки и после того, как она превысит секретный ключ p, расшифровать число становится невозможным. В качестве решения этой проблемы Крэйгом Гентри была предложена техника, называемая bootstrapping, которая, впрочем, приводит к очень быстрому росту объема данных. Как следствие, практическое использование такого шифрования невозможно.

2.2. 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 $a(x), b(x) \in \mathbb{Z}[x]$:

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

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:

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

Из данных формул видно, что свободный член результирующего полинома будет равен сумме и произведению свободных членов исходных полиномов соответственно.
В качестве решения задачу построения отображения из кольца $\mathbb{Z}$ в кольцо полиномов $\mathbb{Z}[x]$ выберем следующий алгоритм:

Алгоритм 2.1. 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 2.2. Сопоставление целого числа полиному (обратный алгоритм для 2.1).

1) Пусть дан полином $g_z(x)$, полученный в результате выполнения алгоритма 2.1.
2) Тогда $g_z(x_0) = q^n f(x_0) - q^n f\left(\frac{p}{q}\right) + z = z$ – исходное число.

Утверждение 2.1. Пусть $h: \mathbb{Z} \to \mathbb{Z}[x]$ – отображение, построенное согласно алгоритму 2.1. Тогда данное отображение является инъективным (разнозначным).

Доказательство следует очевидным образом из случайного выбора коэффициентов.

2.3. Построение гомоморфизма из кольца Z в кольцо полиномов

Теперь рассмотрим задачу построения гомоморфизма из кольца $\mathbb{Z}$ в кольцо $\mathbb{Z}[x]$.

Определение 5 (Гомоморфизм колец)**.** Пусть $f: A \to B$, где A и B – кольца со сложением, умножением, нулем и единицей. Тогда f – гомоморфизм колец, если выполняются следующие условия:

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

Теорема 2.2. Пусть даны кольцо целых чисел $\mathbb{Z}$, кольцо полиномов $\mathbb{Z}[x]$, отображение $P(x)$ построено согласно алгоритму 2.1. Тогда $P(x)$ является гомоморфизмом $P: \mathbb{Z} \to \mathbb{Z}[x]$.

Доказательство. Пусть $z_1, z_2 \in \mathbb{Z}$. Покажем, что выполняется свойство (1), т.е. $P(z_1 +_Z z_2) = P(z_1) +_{Z[x]} P(z_2)$.

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

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

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

Основываясь на выкладках, приведенных в пунктах 2.2 и 2.3, можно построить алгоритмы шифрования и дешифрования. Далее будет приведено две такие схемы.

2.3. Схема 1

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

Алгоритм 2.3 (шифрования)**.** 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 $a_0, a_1, \dots a_n \in \mathbb{Z}$ 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) Тогда шифрующий полином для z будет выглядеть следующим образом:

   $g_z(x) = \ q^n f(x) - q^n f\left(\frac{p}{q}\right) + z \in \mathbb{Z}[x]$.

Алгоритм 2.4 (дешифровки)**.** Рассмотрим алгоритм дешифровки $Dec: \mathbb{Z}[x] \to \mathbb{Z}$, обратный алгоритму 2.3.

1) Пусть полином $g''_{z'}(x) \in \mathbb{Z}[x]$ – зашифрованные данные, $x_0 \in \mathbb{Q}$ – 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 теореме 2.2, данное шифрование является гомоморфным, то есть для $z_1, z_2 \in \mathbb{Z}$ выполняются следующие свойства:

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

В данной криптографической схеме степень n полинома $g_z(x)$ является общедоступным параметром.

2.4. Схема 2

Для построения второй криптографической схемы, воспользуемся китайской теоремой об остатках для полиномов.

Теорема 2.3 (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.

В схеме воспользуемся следующим алгоритмом сопоставления полинома целому числу:

Алгоритм 2.5. 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) 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 (теореме 2.3).

Теорема 2.4. Пусть дано кольцо целых чисел $\mathbb{Z}$, кольцо полиномов $\mathbb{Z}[x]$. Тогда отображение $P(x)$, построенное согласно алгоритму 2.5, 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. ■

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

Алгоритм 2.6 (шифрования)**.** 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 $u(x_0) = 0$. К примеру, если $x_0 = \sqrt{2} + \sqrt{3}$, то полином $u(x)$ можно построить 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 – произвольно выбранное алгебраическое число.
4) Построим шифрующий полином $g_z(x)$ как отображение числа z согласно алгоритму 2.5 с использованием полинома $u'(x) = u(x + c)$, получившегося после замены переменной.

Алгоритм 2.7 (дешифровки)**.** Рассмотрим алгоритм дешифровки $Dec: \mathbb{Z}[x] \to \mathbb{Z}$, обратный алгоритму 2.6.

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

Данную схему можно рассматривать как обобщение схемы 1 из пункта 2.4.

2.5. Реализация алгоритмов в приложении

В приложении «Защищенный калькулятор» данный алгоритмы применяется следующим образом.

Пусть клиенту требуется вычислить функцию $f(x_1, \dots x_m) \in \mathbb{R}$, содержащую операции сложения, умножения и вычитания в точке $z = (z_1, \dots, z_m)$, где $z_i \in \mathbb{Z}$. Для того, чтобы выполнить вычисление, последовательно выполняются следующие шаги:

1) Генерируется закрытый ключ x_0.
2) Согласно одному из алгоритмов выбираются m шифрующих полиномов $g_1, \dots g_m$:

   Для схемы 1:

   $$g_1(x_0) = a_{00} + a_{01}x_0 + a_{02}x_0^2 + \dots + a_{0n}x_0^n = z_1$$
   $$g_2(x_0) = a_{10} + a_{11}x_0 + a_{12}x_0^2 + \dots + a_{1n}x_0^n = z_2$$
   $$\dots$$
   $$g_m(x_0) = a_{m-1\,0} + a_{m-1\,1}x_0 + a_{m-1\,2}x_0^2 + \dots + a_{m-1\,n}x_0^n = z_m$$

   Для схемы 2:

   $$g_1(x) = s_1(x)u(x) + z_1$$
   $$g_2(x) = s_2(x)u(x) + z_2$$
   $$\dots$$
   $$g_m(x) = s_m(x)u(x) + z_m$$

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 вычисление функции $f(x_1, \dots x_m)$ над полиномами $g_1(x), \dots g_m(x)$, т.е. $Res(x) = f\big(g_1(x), \dots g_m(x)\big)$.
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 криптостойкости алгоритмов

В рамках данной дипломной работы помимо разработки схемы полностью гомоморфного шифрования требовалось провести анализ устойчивости шифрования. Для этого исследуем его на предмет устойчивости к следующим видам атак:

- полный перебор
- атака с подобранным открытым текстом
- атака с подобранным зашифрованным текстом

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 секретным ключом является рациональное число $x_0 = \frac{p}{q} \in \mathbb{Q}$. Таким образом, для реализации атаки методом полного перебора требуется перебрать все возможные пары целых чисел (p, q). Количество чисел в машинном изображении множества целых чисел зависит реализации и используемых библиотек, то есть ограничено объемом оперативной памяти. В качестве примера, в языке Java, использовавшемся для реализации алгоритмов, целочисленный тип long позволяет хранить 2^{64} различных значений. В таком случае атака полным перебором не представляется возможной.

Для схемы 2 $x_0 \in \mathbb{A}$. В данном случае ситуация аналогична: представление алгебраических чисел в используемых библиотеках также ограничено размером оперативной памяти.

В реализации этой схемы (см. главу 3) для генерации секретного ключа использовался пакет java.math. В этом пакете есть метод, позволяющий генерировать псевдослучайные вещественные числа типа double. Так как этот тип имеет размер в 8 байт, то существует около 2^{64} различных значений секретного ключа. В таком случае перебрать все варианты ключа также не представляется возможным. Следовательно, данная схема признается устойчивой к атаке полным перебором.

Рассмотрим атаки с подобранным открытым текстом и шифротекстом. Ниже приведен перевод определения данных атак, предложенный Венбо Мао[3].

Определение 6 (Атака на основе подобранного текста)**.** Атакующий выбирает исходные сообщения и передает их шифровальщику для получения зашифрованных

текстов. Задача атакующего – взломать криптосистему, используя полученные пары открытых и зашифрованных текстов.

Определение 7 (Атака на основе подобранного зашифрованного текста)**.** Атакующий выбирает зашифрованные сообщения и передает их на расшифровку для получения исходных сообщений. Цель атакующего – взломать криптосистему, используя полученные пары открытых и зашифрованных текстов. Атакующий достигает успеха, если он раскрывает ключ и способен в дальнейшем извлекать секретную информацию из зашифрованного текста, не прибегая к посторонней помощи.

К сожалению, данные виды атак могут быть успешно применены к обеим криптографическим схемам.

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

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

2.7. Оценка сложности операций в криптосистеме

Произведем оценку алгоритмической сложности выполнения операций сложения и умножения над зашифрованными данными. 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 соответственно.

Тогда очевидным образом 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 соответственно, $p(x)$ – полином замены степени $l > 0$. Тогда операция сложения будет иметь сложность $O(l(n + m))$, умножения – $O(nml^2)$, то есть в случае $l = 1$ сложность операций – $O(n + m)$ и $O(nm)$ – аналогична сложности для первой схемы.

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. Реализация клиентской компоненты приложения «Защищенный калькулятор»

Реализация предложенных алгоритмов является важной частью работы, так как является единственной возможностью на практике оценить их эффективность. В разделе 1.2. 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 состоит из двух частей: клиента и сервера. Рассмотрим их более подробно:

3.1. Клиентская компонента

Пользователь клиентской компоненты задает функцию от m переменных и m-мерную точку, в которой требуется вычислить значение данной функции. Функция может содержать операции сложения, вычитания, произведения, деления, возведения в степень, в том числе, со скобками.

После задания функции случайным образом выбирается ключ шифрования, происходит шифрование исходных данных с помощью полиномов согласно алгоритму и передача по каналу связи на сервер.

После получения ответа от сервера, результат расшифровывается и выдается пользователю.

3.1.1. Генерация секретного ключа

В качестве секретного ключа реализованной криптографической системы из пункта 2.5 выступает случайное вещественное число. Для того чтобы обеспечить максимальную криптографическую устойчивость алгоритма, нужно минимизировать вероятность подбора этого ключа.

Так как невозможно добиться его равновероятностного выбора на промежутке от $-\infty$ до $+\infty$, то предполагается использовать один из следующих подходов:

1. Воспользоваться одним из непрерывных распределений вероятности, например, нормальным распределением. В таком случае, выбор ключа не будет равновероятным, однако он будет выбираться из всего промежутка $(-\infty; +\infty)$, а значит, предсказать зашифрованное значение без знания секретного ключа сложнее.

   Пример.

   Предположим, что ключ выбирается не на $(-\infty; +\infty)$, а на промежутке $(0; 1)$. В таком случае, потенциальному злоумышленнику даже без использования каких-либо криптографических атак будет известно, что число, после шифровки переведенное в полином $x^3 + 3x - 2$ принимает значение от -2 до 2:

$$\min_{x\in(0;1)} (x^3 + 3x - 2) = -2$$
$$\max_{x\in(0;1)} (x^3 + 3x - 2) = 2$$

   В ситуации, когда ключ выбирался на $(-\infty; +\infty)$, злоумышленник никакой дополнительной информацией не обладает:

$$\min_{x\in(-\infty;+\infty)} (x^3 + 3x - 2) = -\infty$$
$$\max_{x\in(-\infty;+\infty)} (x^3 + 3x - 2) = +\infty$$

2. Воспользоваться равномерным распределением. Оно также, как и в первом случае является непрерывным, но принципиальное отличие от предыдущего пункта в том, что ключ выбирается из ограниченного промежутка $[a; b]$, а плотность вероятности на всем этом интервале постоянна.

   Первый факт приводит к потенциальной бреши в криптографической защите алгоритма, как было показано ранее в примере. В качестве частичного решения этой проблемы предлагается выбирать промежуток $[a; b]$ достаточно широким для того, чтобы после шифрования некоторого числа s получившийся полином $\varphi(s)$ на этом промежутке также принимал широкий диапазон значений, и тем самым давал потенциальному злоумышленнику как можно меньше информации.

Из того, что плотность вероятности на всем интервале $[a; b]$ постоянна, следует, что выбор ключа на этом промежутке будет проходить равновероятно. Иными словами, вероятность ключа быть в произвольном промежутке $[c; d] \subset [a; b]$ не зависит от конкретных чисел c и d, а только от их разницы, и равна $\frac{d-c}{b-a}$. Это усложнит процесс подбора ключа к криптографической системе, а значит, повысит её устойчивость к этому виду атаки.

Для реализации был выбран второй подход к выбору ключа, основной причиной для этого является наличие в языке Java встроенной библиотеки java.math, реализующей, в числе прочего, выбор псевдослучайного числа, подчиняющийся равномерному непрерывному распределению.

3.1.2. Графический интерфейс

Для реализации графического интерфейса использовалась библиотека Swing, разработанная компанией Sun Microsystems. Основными причинами выбора этой библиотеки являются:

- богатый набор интерфейсных примитивов;
- встроенная поддержка HTML;
- легкость освоения библиотеки;

Так как основной задачей приложения с точки зрения клиента является выполнение вычислений, для достижения простоты и удобства интерфейса, внешний вид клиентской компоненты был максимально приближен к аналогичному по функциональности интерфейсу стандартного калькулятора Windows 7 в режиме «Обычный».

3.2. Серверная компонента

Сервер представляет собой модель облачного сервиса, предоставляющего клиентам ресурсы, такие как оперативная память, дисковое пространство, процессорное время и т.д.

Серверная компонента должна выполнять заданные операции над зашифрованными данными без их предварительной расшифровки (так как ключ шифрования находится у клиента, расшифровать данные на сервере не представляется возможным).

На сервере вычисляется функция над зашифрованными данными. После вычисления результат (в виде полиномиальной дроби) отправляется обратно клиенту.

3.3. Библиотека работы с полиномами

Поскольку все вычисления на сервере происходят над полиномами, а клиентская компонента выполняет над ними функции шифровки и дешифровки, было решено написать библиотеку для работы с полиномами.

Так как типы double и float могут принимать лишь ограниченное число значений и не являются с математической точки зрения не только алгебраическими, но даже вещественными числами, использование их в качестве коэффициентов полинома повлекло бы за собой ряд ошибок, связанных с неточностью вычислений.

Для решения этой проблемы, в реализации в качестве коэффициентов полинома было решено использовать объекты класса java.math.BigDecimal, реализующего длинную арифметику, а значит позволяющего избежать ошибок, связанных с округлением. Наличие подобной библиотеки является одним из важных факторов, по которым для реализации был выбран язык Java (см. 1.2).

Чтобы удовлетворять всем требованиям системы, класс полиномов Polynomial должен реализовывать следующие действия:

- создавать полиномы с заранее заданными коэффициентами;
- производить над полиномами простейшие математические операции, такие как сложение, вычитание, умножение, а так же умножение на константу;
- выводить значение коэффициента при любой заданной степени переменной, в том числе нулевого;
- менять значение коэффициента при любой степени на любое заранее заданное значение;
- выводить текущие значения коэффициентов полинома в удобном для восприятия человеком виде, например в таком: $x^3 - 1.5\,x + 7$;
- вычислять значение полинома в заданной точке;

Для реализации операций сложения и умножения были использованы классические алгоритмы, работающие за время $O(n)$ и $O(n^2)$ соответственно (операция вычитания полностью аналогична операции сложения), так как они имеют простую реализацию и высокий потенциал для распараллеливания. Операция умножения на константу также работает за $O(n)$. Для вычисления значения полинома в точке используется схема Горнера[4], позволяющая минимизировать число операций умножения и вычисляющая искомое значение за $O(n)$.

Чтобы добиться возможности выполнения на сервере операций деления над зашифрованными данными, было решено использовать вместо полиномов полиномиальные дроби, т.е. дроби, в которых числителем и знаменателем являются некоторые полиномы.

В свою очередь, класс полиномиальных дробей PolynomialFraction должен уметь выполнять следующие действия:

- создавать полиномиальные дроби с заранее заданным числителем и знаменателем из класса Polynomial;
- создавать полиномиальную дробь как результат операции деления между полиномами;
- не позволять создавать полиномиальные дроби с нулевым знаменателем;
- производить над полиномиальными дробями простейшие арифметические операции, такие как сложение, вычитание (с автоматическим приводом к общему знаменателю в случае необходимости), умножение и деление;
- выводить текущие значения числителя и знаменателя полиномиальной дроби в удобном для восприятия человеком виде, например в таком: $\left(3.4\,x^3 - 2.14\,x^2 + 3\,x - 7\right)/(x^2 + 2.5\,x + 1)$;
- вычислять значение полиномиальной дроби в любой заданной точке, при условии, что знаменатель в ней не обращается в ноль;
- преобразовывать полиномиальную дробь в полином в случае, когда знаменатель является ненулевой константой (полиномом нулевой степени) путем деления числителя на эту константу;

Как было сказано ранее, обе части приложения «Защищенный калькулятор» - как клиент, так и сервер – используют данную библиотеку. А потому обеспечение корректности её работы является важной задачей для разработки всей экспериментальной реализации. 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 Java существует несколько инструментов и библиотек:

- JUnit
- TestNG
- JavaTESK

Для тестирования библиотеки была выбрана библиотека JUnit, как наиболее популярная, простая, и в то же время обладающая 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. Тестирование производительности

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

Для тестирования производительности было решено создать аналог существующей криптографической библиотеки на языке C++ с использованием длинной арифметики библиотеки GMP, поскольку язык Java значительно уступает C++ в вычислительной производительности.

В работе использовалось два вида тестов производительности:

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 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 (p, q) 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 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

В таблице 1 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.

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.

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.

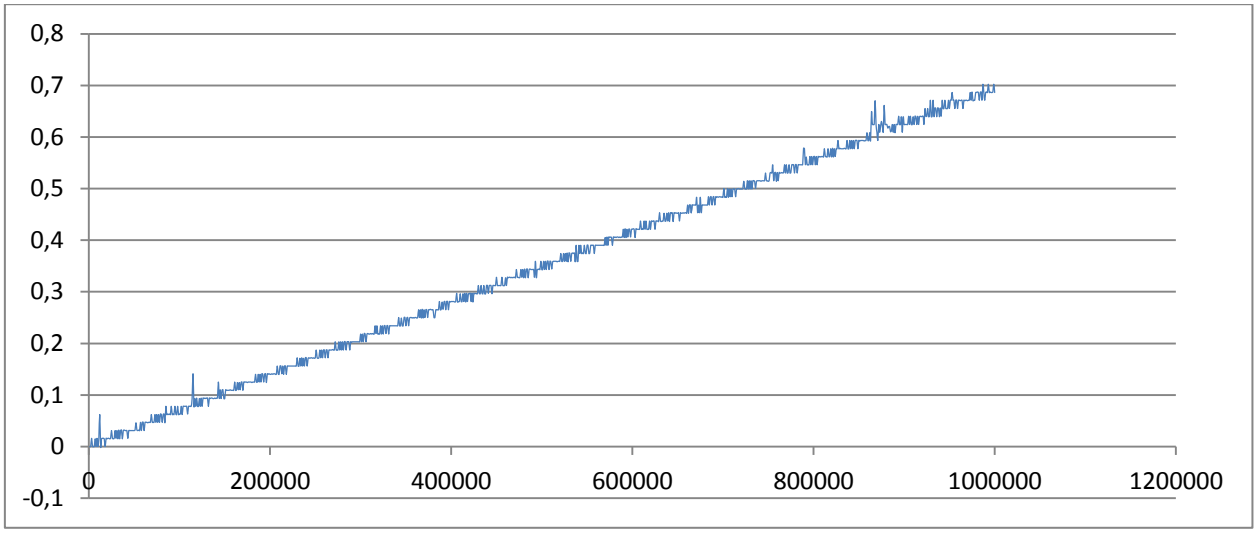

Рис. 1. График зависимости времени, приходящегося на одну операцию сложения, от количества операций в серии

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 – $O(n^2)$.

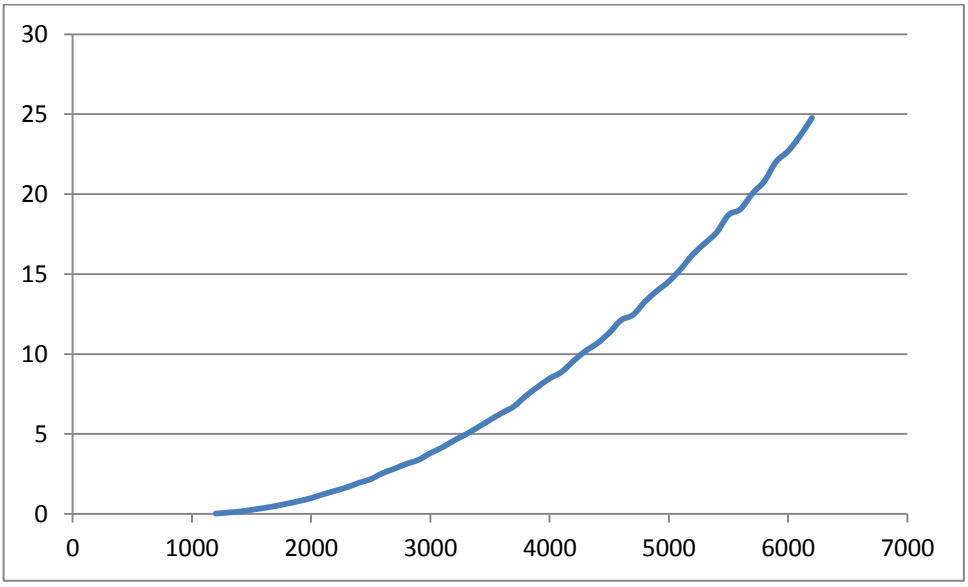

Рис. 2. График зависимости времени, приходящегося на одну операцию умножения, от количества операций в серии

Заключение

Разработанная схема полностью гомоморфного шифрования имеет большие перспективы использования, так как выгодно отличается от ранее предложенных алгоритмов. Тем не менее, у неё имеется существенный недостаток - повышение потребления памяти и потеря производительности с ростом числа выполненных умножений над полиномами. Кроме того, данная схема имеет высокий потенциал ускорения за счет распараллеливания вычислений.

Главной сферой использования данной схемы являются облачные вычисления. Кроме того, результаты данного исследования планируется использовать для создания исполняемого кода, защищенного от реверс-инжиниринга.

Таким образом, существует несколько возможных направлений развития:

- устранение основного недостатка схемы – рост степени полиномов с ростом числа выполненных умножений над данными (например, с помощью фактор-кольца многочленов)
- улучшение устойчивости схемы к различным видам криптоанализа, например использование полиномов от нескольких переменных
- использование графических ускорителей для повышения производительности вычислений
- использование полученных результатов для создания защищенной СУБД

Список литературы

1. Mell, Peter. The NIST Definition of Cloud Computing / Peter Mell, Timothy Grance. -2001. – Sept. http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf
2. Rivest, R.L. On data banks and privacy homomorphism / R.L. Rivest, L. Adleman, M.L. Dertouzos // Foundations of secure computation. -1978. –Vol. 32, no. 4. –Pp. 169-178. http://people.csail.mit.edu/rivest/pubs/RAD78.pdf.
3. Mao, Wenbo. Modern Cryptography. Theory and practice. / Wenbo Mao, Hawlett-Packard Company. – М.: Издательский дом «Вильямс», 2005.
4. 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.
5. Шнайдер, Б.А. Прикладная криптография. – Москва: Триумф, 2002. -816 с.
6. Гомоморфное шифрование. Н.П. Варновский, А.В. Шокуров //Труды Института Системного программирования: Том 12. (под Ред. В.П. Иванникова) – М.:ИСП РАН, 2006.
7. Скляров, Д. Искусство защиты и взлома информации. – СПб: БХВ-Петербург, 2004, -271 с.
8. Craig Gentry. A fully homomorphic encryption scheme // PhD thesis. – Stanford University. – 2009. – URL: http://crypto.stanford.edu/craig