Санкт-Петербургский государственный университет

**Факультет прикладной математики – процессов управления**

**Кафедра Технологии Программирования**

**Корчагин Дмитрий Дмитриевич**

**Дипломная работа**

**Обеспечение надежности хранения данных в клиент-серверных системах**

Заведующий кафедрой,  
кандидат физ.-мат. наук,  
доцент Сергеев С. Л.

Научный руководитель,  
ст. преподаватель Севрюков С. Ю.

Рецензент,  
доктор физ.-мат. наук,  
профессор Фамилия И. О.

Санкт-Петербург

2014

Оглавление

[Введение 4](#_Toc389441090)

[Терминология 10](#_Toc389441091)

[Постановка задачи 11](#_Toc389441092)

[Глава 1. Определение требований и выбор оптимального решения. 12](#_Toc389441093)

[1.1. Исследование бизнес-процессов. 12](#_Toc389441094)

[1.2. Проблемы 13](#_Toc389441095)

[1.3. Требования к безопасности данных 14](#_Toc389441096)

[1.4. Существующие сервисы 15](#_Toc389441097)

[Dropbox 16](#_Toc389441098)

[Waula 16](#_Toc389441099)

[SpiderOak 17](#_Toc389441100)

[UbuntuOne 17](#_Toc389441101)

[1.5. Существующие решения 17](#_Toc389441102)

[Boxcryptor 17](#_Toc389441103)

[Viivo 18](#_Toc389441104)

[Cloudfogger 19](#_Toc389441105)

[EncFS 19](#_Toc389441106)

[1.6. Разработка собственного решения 20](#_Toc389441107)

[1.7. Выводы 21](#_Toc389441108)

[Глава 2. Разработка модуля шифрования. 22](#_Toc389441109)

[2.1. Основная концепция и выбор технологий 22](#_Toc389441110)

[2.2. Архитектура разрабатываемого решения 22](#_Toc389441111)

[2.3.Определение сущностей разрабатываемого решения 26](#_Toc389441112)

[2.4.Алгоритм работы модуля 28](#_Toc389441113)

[2.5. Достигнутые результаты 30](#_Toc389441114)

[2.6.Тестирование разработанного решения 31](#_Toc389441115)

[Глава 3. Результаты 35](#_Toc389441116)

[3.1. Дальнейшее развития модуля 35](#_Toc389441117)

[Заключение 37](#_Toc389441118)

[Список литературы и источников 38](#_Toc389441119)

[Приложение 1. Графики на основе полученных в ходе тестирования данных. 39](#_Toc389441120)

[Приложение 2. Исходный код заголовочных файлов реализованных модулей 43](#_Toc389441121)

[Приложение 3. Скриншоты работы программы 50](#_Toc389441122)

[Приложение 4. Рекомендуемые энциклопедические статьи. 54](#_Toc389441123)

# Введение

На заре развития ЭВМ информации было относительно мало, она передавалась медленно и обладала гораздо меньшей ценностью, чем сейчас. Для того чтобы в этом убедиться, достаточно обратить внимание на ведущиеся сейчас информационные войны. Когда-то вычисления проводились вручную, что занимало достаточно много времени, кроме того существовала огромная вероятность появления ошибок, проверка же результата была столь же затратна по времени, как и сами вычисления. Затем появились ЭВМ - они обладали возможностью производить вычисления быстро и точно, однако в те времена они занимали огромные площади и стоили очень дорого, позволить их себе могли лишь научно-исследовательские центры и крупные компании. Поскольку необходимо было обмениваться информацией (и как можно быстрее) был создан Интернет. Однако большинство людей не нуждалось в проведении сложных вычислений (а это был единственный способ использования ЭВМ), а значит не было необходимости в персональных компьютерах.

Со временем компьютеры и их стоимость становились все меньше, а их возможности и область использования расширялись. Так, уже 5 лет назад, компьютер был в большинстве домов. Средства обмена данными также совершенствовались - при росте объема данных потеряли популярность дискеты (стандартный объем которой составлял 1.44Мб), а затем начали выходить из обихода и компакт-диски. Эти носители информации были относительно неудобны, их легко было повредить. На смену пришли flash-накопители, они ещё пользуются популярностью, хотя их роль становится всё менее значимой при ежедневном обмене информацией.

В современном мире Интернет играет очень важную роль: с помощью него мы общаемся с друзьями, обмениваемся данными, играем, а так же храним данные и находим требующуюся нам информацию. Компьютеры стали меньше, наступила эра мобильных приложений. Мобильность играет всё большую роль, использование компьютера для просмотра веб-страниц кажется уже неоправданным, для этого гораздо удобнее взять планшет, который всегда может быть рядом с вами. С другой стороны, услуги, предоставляемые различными компаниями, теперь уже располагаются не на личных компьютерах пользователей - всё чаще они располагаются на удаленных серверах, доступ к которым осуществляется с помощью глобальной сети.

В ближайшем будущем развитие вычислительных устройств будет и дальше идти в сторону мобильности. В сторону развития устройств, которые всегда будут под рукой, а вместе с тем и беспроводных средств связи, позволяющих в любое время обмениваться информацией. В качестве примера можно привести разработки крупнейших IT-гигантов Google Glass, iWatch, которые стремятся перехватить инициативу на новом витке развития мобильных технологий. Поскольку мобильные устройства не обладают достаточными мощностями для осуществления сложных вычислений, и объем их памяти довольно мал по сравнению со стационарными компьютерами, всё больший объем вычислений производится на удаленных серверах, а личные данные при необходимости получают из глобальной сети с помощью систем синхронизации. Однако, при обмене личными данными через Интернет, следует подумать об их защищенности (информация не должна быть доступна третьим лицам без Вашего разрешения).

Впервые о разработке облачных технологий задумались еще на заре развития сети Интернет. Идея использования удаленных вычислительных мощностей не только для хранения данных, но и для средств их обработки появилась в середине 70-ых годов, однако вплоть до 90-ых развитие облачных технологий сдерживалось относительной неразвитостью сети Интернет, в частности низкой пропускной способностью каналов.

Однако в дальнейшем произошёл прорыв, и облачные сервисы демонстрировали свое поступательное развитие вместе с ростом масштабов Всемирной Сети.

Облака, по сути, представляют из себя удаленные сервера, которые предоставляют свои мощности для конечных пользователей. С клиента снимается ответственность по администрированию и настройке машин, эти задачи перенимает на себя хостинг-провайдер. Он же обеспечивает непрерывный доступ пользователя к серверам. Сам же пользователь платит только за использование этих серверов.

*Облачные сервисы* - модель обеспечения повсеместного сетевого доступа к общему пулу конфигурируемых вычислительных ресурсов, которые могут быть оперативно предоставлены и освобождены с минимальными эксплуатационными затратами.

Потребители облачных сервисов могут уменьшить расходы на инфраструктуру информационных технологий и гибко реагировать на изменения вычислительных потребностей, используя свойства вычислительной эластичности облачных услуг. Под эластичностью понимается запуск и подключение дополнительных узлов в случае превышения оптимальной нагрузки на используемые ресурсы.

Облачных вычисления можно разделить на три группы:

1. Сервис облачных приложений
2. Сервис облачных вычислений
3. Сервис облачного хранения

*Облачные приложения*

Так называемые SaaS (software as a service) - программное обеспечение, предоставляемое как услуга. Является бизнес-моделью продажи и использования программного обеспечения, при которой поставщик разрабатывает веб-приложение и самостоятельно управляет им, предоставляя клиенту доступ к программному обеспечению через Интернет. Основное преимущество этой модели для потребителя - отсутствие затрат, связанных с установкой и поддержкой работоспособности оборудования и работающего на нем программного обеспечения.

В модели SaaS:

* приложение приспособлено для удаленного использования
* одним приложением пользуется множество клиентов
* оплата происходит либо в виде ежемесячной платы, либо на основе объема операций
* техническая поддержка сервиса включена в оплату
* обновление приложения происходит прозрачно для клиента.

*Облачные вычисления*

При облачных вычислениях хостинг-провайдер (владелец вычислительных машин) предоставляет только вычислительные машины, позволяя клиентам загружать и запускать на них собственные (разработанные клиентом) сервисы. При этом провайдер обеспечивает бесперебойную работу вычислительных машин, так же обеспечивается непрерывный доступ к сервису как для конечных клиентов, так и для владельцев сервиса, в обязанности которых входит обновление и обеспечение стабильности загруженных ими приложений.

*Облачное хранение*

Модель онлайн-хранилища, в которой данные хранятся на многочисленных распределенных в сети серверах, предоставляемых в пользование клиентам. В отличие от хранения данных на собственных выделенных серверах (приобретаемых или арендуемых для подобных целей) внутренняя структура серверов клиенту не видна. Данные хранятся в так называемом “облаке”, которое воспринимается клиентом как единственный виртуальный сервер, хотя физически эти сервера могут располагаться как угодно.

*Облачные хранилища*

Как уже говорилось, в современном мире редкий человек пользуется лишь одной вычислительной машиной. В то же время есть необходимость в постоянном доступе к личным или рабочим данным и их синхронизации на различных устройствах. Это стало возможно благодаря развитию сети Интернет. Всё больше устройств подключено к глобальной сети, а значит, появляется возможность удаленной передачи данных между этими устройствами. Принципиально новым способом передачи и хранения данных стали облачные хранилища - удаленные сервера, предоставляющие возможность хранения на них личных данных, а также возможность получения этих данных с любого устройства, подключенного к сети.

Одним из первых сервисов подобного рода стал Dropbox - хранилище, принадлежащее компании Dropbox Inc. Считается, что идея создания подобного сервиса родилась у основателя компании Дрю Хьюстона во время автобусной поездки, когда он забыл флешку со своими данными.

Dropbox позволяет размещать файлы на удаленных серверах с помощью клиента или веб-интерфейса (через браузер). При использовании клиента создается специальная папка, данные в которой синхронизируются с облаком - это является основным принципом синхронизации данных с помощью облачных хранилищ.

На данный момент существуют десятки сервисов, позволяющих хранить свои данные на удаленных серверах. В основном они различаются скоростью соединения и предоставляемым объемом дискового пространства, которое доступно пользователю. Но существует и ряд более тонких отличий, таких как возможность открытия доступа к части своих данных конкретному лицу или всем пользователям (открытый доступ), возможность шифрования данных при хранении, проведение хостингом определенных действий над хранимыми клиентом данными.

# Терминология

Во избежание путаницы есть необходимость определить следующие термины, для их однозначной трактовки в рамках работы.

* *сервис* - предоставляемая услуга (хранение, предоставление мощностей или разворачивание приложения)
* *серверная сторона* - кластер удаленных машин, на которых осуществляется выполнение сервиса.
* *клиент* - приложение, работающее на стороне пользователя (фронт-энд приложения)
* *пользователь* - потребитель сервиса
* *провайдер* (*хостинг*) - владелец кластера удаленных машин
* *локальная машина -* компьютер (или несколько компьютеров), находящиеся под контролем пользователя.

# Постановка задачи

Целью данного исследования является изучение принципа работы облачных сервисов, а так же анализ и решение проблем, возникающих при шифровании синхронизируемых данных на стороне клиента. Для достижения поставленной цели предлагается решить задачу разработки модуля, отвечающего за шифрование данных до их передачи в облако и обладающего интерфейсом, который позволил бы использовать его в клиентах для облачных сервисов.

Решение данной задачи требует реализации следующих этапов:

1. Исследование предметной области
   1. Анализ бизнес-процессов и определение потребностей пользователя
   2. Изучение существующих сервисов облачной синхронизации
   3. Изучение существующих решений поставленной задачи
2. Формирование требований к программному продукту
3. Анализ и обоснование выбора методов решения
4. Разработка модуля:
   1. Выбор технологий
   2. Разработка архитектуры
   3. Реализация модулей
5. Тестирование разработанного модуля

# Глава 1. Определение требований и выбор оптимального решения.

В данной главе будет исследована предметная область и проведен анализ бизнес-процессов, протекающих в этой области. Также будут сформулированы требования к разрабатываемому решению и выбран оптимальный подход, позволяющий покрыть необходимый функционал.

## 1.1. Исследование бизнес-процессов.

Основным объектом исследуемой области являются данные. Они создаются, изменяются и передаются от одного участника процесса к другому (в качестве участников могут выступать как различные клиенты на разных машинах, принадлежащие одному пользователю - так и клиенты, принадлежащие разным пользователям). Примером может служить схема, при которой файлы закачиваются на какой-либо ресурс, а затем скачиваются с этого ресурса на другую машину.

В рамках работы были исследованы основные задачи, решаемые пользователями при использовании облачных сервисов. Ниже представлены некоторые из них:

* резервное копирование данных;
* синхронизация данных между различными устройствами;
* возможность доступа к данным для третьих лиц;
* совместный доступ к данным.

Существует множество сервисов, предлагающих решения для поставленных выше задач. Однако, большинство из них обладают некоторыми недостатками.

## 1.2. Проблемы

При использовании облачных хранилищ возникает несколько проблем для пользователя.

1. Могут возникнуть трудности при установке клиента. То, что продвинутые пользователи ПК считают очевидным - к примеру, поиск файла в древовидном каталоге жесткого диска - для некоторых пользователей может оказаться нетривиальной задачей. В таких условиях разрабатываемая утилита должна быть максимально удобной, и минимизировать дополнительные усилия, которые придется приложить пользователю для его использования
2. Для доступа к актуальным данным необходимо подключение к Интернету, и скорость получения данных напрямую зависит от пропускной способности канала
3. Хотя владельцем данных и является пользователь - однако, все данные полностью подконтрольны провайдеру. А это значит, что пользователь не контролирует целостность и конфиденциальность своих данных

*В частности, недавний скандал с облаком от mail.ru, когда выяснилось, что в лицензионном соглашении были прописаны пункты, дающие право компании Mail.ru Group пользоваться на своё усмотрение всеми данными, которые пользователи размещали на их серверах.*

Для провайдера же основной проблемой является поддержка стабильной работы внутренней архитектуры и обеспечение доступа к данным в любое время. Также владелец серверов должен обеспечивать безопасность учетных записей клиентов и сохранность их данных. В остальном работа провайдера сводится к минимизации усилий, прикладываемых потребителями для пользования предоставляемыми услугами (предоставление максимально удобных клиентов).

## 1.3. Требования к безопасности данных

Основной проблемой для клиента является невозможность контролировать свои данные, в то время как они находятся у хостинг-провайдера. Несмотря на лицензионные соглашения и гарантии, нельзя быть полностью уверенным в их сохранности и сохранении конфиденциальности - даже в случае, когда хостинг-провайдер не использует полученные им данные, возможная утечка данных компании злоумышленнику может раскрыть и ваши файлы.

В качестве решения предлагается разработать утилиту, позволяющую производить шифрование данных до их отправки в облако, и оценить, насколько трудозатратно было бы обеспечивать безопасность данных таким образом при использовании произвольного облачного сервиса.

В ходе анализа требований к утилите были выделены следующие:

* прозрачность;
* отказоустойчивость;
* нагрузка на канал.

*Прозрачность* - для использования утилиты пользователю следует проводить минимальное количество дополнительных операций. В идеале клиент может даже не догадываться о том, в какое время происходит шифрование и синхронизация данных с облачным хранилищем.

*Отказоустойчивость -* способность системы выполнять свои функции при отказе отдельных элементов аппаратуры и неполной доступности данных.

Основной способ увеличить отказоустойчивость системы - это избыточность хранимых данных, которые в случае отказал части сервиса могут быть получены из других источников.

Саму же отказоустойчивость можно определить с помощью следующих параметров:

* коэффициент готовности (время работы/время службы)
* надежность - вероятность отказа на единицу времени.

Говоря об отказоустойчивости (в рамках предлагаемого решения), следует обратить внимание на решение следующих задач:

* конфиденциальность - данные должны быть доступны только лицам, доступ которым разрешен владельцем. Не должно быть возможности по зашифрованным данным восстановить исходные, не зная ключа.
* целостность - информационная значимость данных не должна изменяться без ведома владельца. При шифровании и синхронизации не должно происходить сбоев, связанных с потерей информации.
* доступность - у пользователя всегда должна быть возможность обратиться к своим данным. Продумать методы восстановления ключа и исходных данных.

## 1.4. Существующие сервисы

Существует множество облачных хранилищ, и все они предлагают различные услуги при хранении данных. Для того чтобы выбрать целевую платформу, я предлагаю рассмотреть несколько наиболее популярных хранилищ и определить их следующие свойства:

* насколько решены описанные в предыдущем разделе проблемы
* открыты ли исходные коды клиента
* кроссплатформенность
* современность, актуальность - ведется ли на данный момент развитие проекта
* специфика системы - уникальные особенность и услуги, предоставляемые владельцами хранилища

### Dropbox

* Стандартный процесс установки, данные хранятся как на машинах пользователя - так и на удаленном сервере. В то же время шифрование “происходит” на стороне сервера - теоретически, третьи лица могут получить к ним доступ
* Закрытый исходный код (python)
* Windows, Linux, Mac OS, iOS, Android
* Ведется разработка
* Дельта-кодирование, позволяющая хранить историю изменения данных.

### Waula

* Стандартный процесс установки, данные шифруются и разбиваются на фрагменты, которые загружаются на сервер а затем распространяются через сеть на машины пользователей, что повышает отказоустойчивость системы.
* Закрытый исходный код (java)
* Windows, Linux, Mac OS, iOS, Android
* Ведется разработка
* Объем предоставляемого в облаке дискового пространства зависит от объема диска, предоставляемого для использования другими пользователями

### SpiderOak

* Быстрая регистрация и установка
* Закрытый исходных код
* Windows, Linux, Mac OS, iOS, Android
* Ведется разработка
* Данные полностью шифруются на клиенте. Тонкая настройка по резервному копированию. Дедубликация, дельта-кодирование

### UbuntuOne

* Предустановлен на ОС Ubuntu, в остальных системах - быстрая регистрация и установка, не требует предварительной настройки.
* Открытый исходных код (python)
* Linux, Windows, Mac OS, Android, iOS
* Закрыт
* Поддержка прокси, синхронизация контактов, потоковое вещание

## 1.5. Существующие решения

Boxcryptor - приложение для шифрования, оптимизированное для облачного использования. Создает виртуальный диск на компьютере, который позволяет отдельные файлы. Любой файл, скопированный на виртуальный диск, будет автоматически зашифрован перед синхронизацией с облаком. Использует AES256 и RSA.

Поддерживает следующие хранилища:

* Dropbox
* Google Drive
* Yandex Disk
* SpiderOak
* Waula
* Ubuntu One
* множество других

Поддерживаемые операционные системы:

* Windows
* Mac OS X
* Chrome
* iOS
* Android
* Win Phone
* Windows RT
* BlackberryOS

Отсутствует: прозрачность для пользователя. Требуется установка дополнительного приложения, а так же его тонкая настройка.

Viivo - приложение получает на вход файлы, шифрует их и автоматически перемещает в папку для синхронизации с облаком.

Поддерживает следующие хранилища: любые, поскольку шифрования реализовано независимо от облачной синхронизации.

Поддерживаемые операционные системы:

* Mac OS X
* Windows
* iOS
* Android

Отсутствует: прозрачность для пользователя, ограничение функционала на бесплатных версиях.

Cloudfogger - создает виртуальный диск, обеспечивая шифрование хранящихся на нем данных. При обращении к тому, данные расшифровываются налету - прозрачно для пользователя.

Поддерживаемые облачные хранилища: любые, поскольку данные шифруются на уровне файловой системы.

Отсутствует: прозрачность для пользователя, кроссплатформенность.

Поддерживаемые операционные системы:

* Windows Vista
* Windows 7
* Windows XP

EncFS - получает папку-приемник и папку-источник. Каждому файлу в папке-приемник соответствует зашифрованный в папке-источник. Ключ для шифрования хранится вместе с зашифрованными файлами также зашифрованным, для его дешифрования и требуется пароль пользователя.

Поддерживаемые облачные хранилища: любые.

Поддерживаемые операционные системы:

* Linux
* Mac OS X
* FreeBSD

Отсутствует: прозрачность для пользователя, дублирование файлов, никак не облегчается нагрузка на канал при синхронизации с облаком.

## 1.6. Разработка собственного решения

Альтернативным подходом является разработка собственного модуля, который отвечал бы за шифрование передаваемых данных.  
Такое решение должно позволить при минимальном изменении существующей инфраструктуры клиентов позволить шифровать данные, передаваемые в облако.

Конечно, этот подход более трудозатратный, чем выбор из существующих вариантов. Однако он позволит в лучшей степени подстроится под существующие архитектуры.

Разработка программного обеспечения, как правило, включает в себя следующие этапы:

1. анализ требований
2. проектирование
3. реализация
4. тестирование
5. внедрение
6. сопровождения

В рамках текущей задачи последние два пункта могут быть опущены.

# 

## 1.7. Выводы

Подводя итоги по существующим вариантам хранения данных в облаке, гарантируя их конфиденциальность, можно сделать вывод что основной способ - это создания виртуального тома, данные на котором хранились бы в зашифрованном виде. И только при обращении пользователя к данным они будут расшифрованы, что задерживает работу пользователя и повышая нагрузку на его компьютер.

Так же множество различных систем вызывают проблемы совместимости, когда не все решения могут быть использованы с любым облачным сервисом.

В итоге было решено создать собственный модуль, обеспечивающий решение всех поставленных задач.

# Глава 2. Разработка модуля шифрования.

На основе выводов предыдущей главы было решено разработать прототип, позволяющий осуществлять шифрование пользовательских данных и реализующий интерфейсы, необходимые для взаимодействия клиента облачного сервиса с этой утилитой. В основном при реализации прототипа будут рассматриваться проблемы трудозатратности и облегчения нагрузки на канал при синхронизации измененных файлов.

## 2.1. Основная концепция и выбор технологий

Одной из ключевых особенностей разрабатываемого решения является возможность интегрировать его в любой клиент, разрабатываемый на произвольном языке программирования. Поэтому логично для разработки использовать язык, использование которого в других приложениях вызовет наименьшее количество проблем. С другой стороны, поскольку в процессе работы утилиты планируются довольно сложные и объемные вычисления, а одним из ключевых показателей является скорость - выбор уходит в сторону языков с наименьшей абстракцией над операционной системой, а именно с\с++.  
 Использование этих языков позволить максимально эффективно использовать вычислительные ресурсы компьютера, при грамотном использовании гарантируя максимальные показатели производительности и минимальные затраты времени. В то же время, у большинства существующих языков есть библиотеки, позволяющие использовать модули, написанные на указанных языках.

## 2.2. Архитектура разрабатываемого решения

Проанализировав указанные требования и выбрав технологии, необходимо определиться с архитектурой модуля. Для этого предлагаю проанализировать работу облачного хранилища.

Изначально, существует некая папка, за которой постоянное наблюдение со стороны клиента. В этой папке могут происходить следующие события:

* создание документа,
* изменение документа,
* удаление документа.

В зависимости от типа события, вызывается определенные метод клиента, позволяющий провести закачку новый данных на удаленный сервер, провести каким-то образом синхронизацию данных (к примеру, это может быть diff-кодирование или отправка нового файла целиком), либо удаленные файла.

На стороне клиента и на стороне сервера существует файл, хранящий метаинформацию о всех данных, находящихся в отслеживаемой папке. Благодаря этой метаинформации клиент определяет, на какой стороне (серверной либо пользовательской) произошли изменения, и какие именно данные синхронизировать. В частности, на примере Ubuntu One, среди метаданных находится путь к файлу на локальной машине, его идентификатор на сервере, время изменения файла и множество другой информации.

После того, как изменившиеся данные синхронизированы с сервером, остальным клиентам (работающим на других машинах) поступает команда о получении данных с сервера. После чего они аналогичным образом, используя файл с метаданными, синхронизируют данные уже на локальной машине.

Грубо говоря, схему работы облачных сервисов можно представить следующим образом:

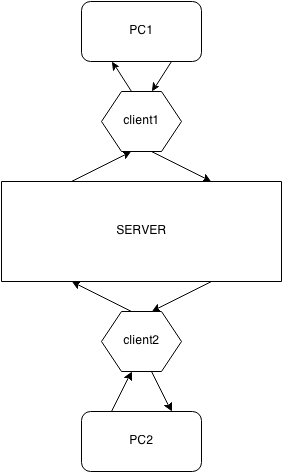


Рис. 1. Схема работы сервисов облачного хранения.

То есть клиент располагается между пользовательской машиной и сервером: с одной стороны он обеспечивает отслеживание изменений на компьютере пользователя, с другой может взаимодействовать с сервером с помощью предоставляемого API.

У разрабатываемого модуля есть три принципиально разных способа воздействия на данные перед тем, как они будут отправлены в облако.

1. Встроить модуль между компьютером пользователя и клиентом
2. Интегрировать разработанный модуль в клиент
3. Встроить модуль между клиентом и сервером.

Рассмотрим подробнее каждый из этих способов:

*Встроить модуль между компьютером пользователя и клиентом.*

Наиболее простой вариант. В этом случае не требуется глубокого анализа исходного кода клиента, достаточно обеспечить прослойку, гарантирующую преобразование (шифрование) файла и генерирующую некий результат, с которым будет работать непосредственно клиент. Конечно, модуль так же должен отслеживать изменения “результата” - для синхронизации данных, с которым взаимодействует пользователь. Такое решение, с точки зрения трудозатрат для пользователя, будет мало отличаться от существующих - однако обладание открытым исходным кодом поможет развить его, предоставив интерфейсы для взаимодействия с клиентом.

*Интегрировать разработанный модуль в клиент.*

Является более сложной и специфичной задачей. В основном это связано с тем, что большинство клиентов имеют закрытый исходный код, и правила взаимодействия с файловой системой (а так же с сетевыми протоколами) нам неизвестны. Кроме того, даже имея исходный код, сложно понять целиком архитектуру приложения - что может привести к усложнению кода и сделать клиента менее расширяемым. Однако, это обеспечило бы максимальную легкость для пользователя, которому не пришлось бы выполнять никаких дополнительных действий, кроме установки клиента на свой компьютер.

*Встроить модуль между клиентом и сервером.*

Наиболее сложный и менее продуктивный вариант - мы не знаем API сервера и правила, по которым он обменивается информацией с клиентом. Кроме того, обработка уже готовых для передачи данных более трудозатратна, чем работа с исходными данными.

В результате анализа представленных вариантов был выбран первый, поскольку это обеспечит лучшую скорость разработки и позволит проанализировать производительность и надежность модуля не учитывая задержек, связанных с работой самого клиента.

## 2.3.Определение сущностей разрабатываемого решения

В соответствии с выбором архитектуры и представленными требованиями, в утилите следует учесть наличие следующий модулей:

* inotify - обеспечивает мониторинг файловой системы, обрабатывает события. Интерфейс обеспечивает возможность указать директорию, события в которой мы будет отслеживать
* filemanager - отвечает непосредственно за взаимодействие с файловой системой.
* scrambler - отвечает непосредственно за шифрование
* config - хранит в себе внутренние настройки программы в ходе её выполнения
* metaworker - обеспечивает работу с метафайлом, в котором хранятся данные необходимые для обработки пришедших с сервера файлов.
* updater - определяет, в какой части файла произошли изменения.
* md5 - стороння библиотека, обеспечивающая взятие хеша произвольной последовательности.

Теперь подробнее о каждой из сущностей:

*inotify*

Является оболочкой для одноименной подсистемы ядра Linux, которая позволяет получать уведомления об изменениях файловой системы. Создается файловый дескриптор, из которого читаются все происходящие события:

*int inotify\_init();*

Внутри этого дескриптора создается уникальный дескриптор наблюдения, который используется для отождествления путей и происходящий событий:

*int inotify\_add\_watch(int fd, const char \*path, \_\_u32 mask);*

При этом отслеживаются только события, который были указаны в маске.

Удаление дескриптора наблюдения:

*int inotify\_rm\_watch(int fd, \_\_u32 wd);*

Позволяет отслеживать следующие события: доступ, изменение атрибутов, закрытия, создание, удаления, модификация, перемещения, открытие.

*filemanager*

Обладает методами, позволяющими обрабатывать только что созданные файлы, проводить обновление уже существующих файлов, либо удалять файлы - а также производить их разбиение на сегменты и склеивание.

*scrambler*

Является оболочкой для открытой библиотеки libcryptopp, реализующей различные криптографические схемы.

В качестве шифрования был выбран AES256 (стандарт симметричного алгоритма блочного шифрования в США), широко использующийся системах различной сложности благодаря своей надежности. Реализует возможность шифрования и дешифрования потока данных, представленных в качестве строки.

*config*

Отвечает за внутренние настройки модуля. К примеру, размер буфера при чтении файла или директория для результирующего набора данных.

*metaworker*

Обеспечивает взаимодействие остальных элементов системы с файлом, содержащим метаданные. В его задачи входит считывание этого файла и представление его в виде, удобном для использования другими компонентами, а также поддержание метаданных в актуальном состоянии при изменении соответствующих им данных.

*updater*

Поскольку передавать весь файл по сети в случае его изменений было бы слишком затратно, был разработан подмодуль, позволяющий определять в какой части файла произошли изменения, чтобы filemanager мог использовать результат для экономии объема трафика (или, в нашем случае, уменьшения объема копируемой информации).

## 2.4.Алгоритм работы модуля

При запуске, устанавливается слежение за двумя директориями: во-первых, за директорией с исходными данными (в дальнейшем буду звать её “исходная”), во-вторых, за директорией которая содержит (или будет содержать) сегментированные копии данных (в дальнейшем “дублированная”). Это необходимо для того, чтобы при синхронизации дублированной папки с удаленным сервером мы могли их расшифровать и синхронизировать с данными из исходной директории.

Далее рассмотрим три возможных ситуации: создание нового файла, изменение существующего, либо удаление.

При создании файла, inotify обрабатывает это событие и вызывает filemanager, который в свою очередь разбивает созданный файл на фрагменты, шифрует эти фрагменты, и сохраняет в новый файл. В то же время filemanager, используя metaworker, создает файл с метаданными, в котором хранится следующие строки (для каждого сегмента используется одна строка):

* md5 от исходного сегмента
* md5 от зашифрованного сегмента
* байт, с которого начиналось считывание сегмента в исходном файле
* байт, которым заканчивалось считывание сегмента в исходном файле.

Именем метафайла является md5 от полного пути к исходному файлу, а название зашифрованных сегментов - md5 от исходного сегмента, что позволяет хранить все зашифрованные сегменты и метафайлы в одном каталоге, не беспокоясь о возможных конфликтах с названиями файлов, содержащимися в исходной папке.

Обратная операция - сбор исходного файла по сегментам, используя файл с метаданными, тривиальна и не требует много пояснений. Зная порядок файлов (благодаря указанию начального и конечного байта) мы по очереди читаем сегменты, расшифровываем их и записываем в исходный файл.

При удалении файла нам достаточно считать соответствующий ему метафайл и удалить сегменты которые в нем упоминались.

При обновлении файла мы вызываем updater. Тот проверяет, совпадают ли хеши считанного сегмента с хешем, сохраненным в метафайле. Если совпадает, мы сравниваем следующий сегмент, и так до конца файла.

Если же хеш не совпадает, мы считываем сегмент, начиная со следующего байта. И так до конца файла, пока не встретим совпадение. Затем сравниваем хеши со следующим сегментом, считая, что первый был изменен. В итоге мы получаем интервалы байтов, зашифрованных копий которых у нас ещё не хранится.

Альтернативным вариантом является проверка всех следующих блоков на совпадение с хешем, начиная с текущего байта, если с первым совпадения не было.

## 2.5. Достигнутые результаты

В ходе создания прототипа модуля шифрования данных при их передачи в облако были достигнуты следующие результаты:

* Рассмотрены протекающие в системе бизнес-процессы
* Определено типичное поведение пользователей, требующее наибольшее внимание
* Выделены критические для пользователей моменты при использовании системы облачного хранения
* Спроектирован прототип, позволяющий определить ресурсы, необходимые для добавление шифрования при условии минимальной нагрузки на канал
* При разработке прототипа реализованы его следующие модули:
  + filemanager - описаны операции, необходимые при добавлении, изменении или удаления файла
  + scrambler - разработаны методы, позволяющие осуществлять шифрование и дешифровку произвольного типа данные с использованием типа шифрования AES256
  + metaworker - реализованы методы, необходимые для оперирования с метаданными:
    - загрузка и распознавание данных
    - получение доступа к произвольным элементам метаданных
    - вставка, замена и удаление элементов
    - изменение произвольного участка метаданных
    - сохранение модифицированных метаданных
  + updater - реализованы методы, позволяющие определить оставшиеся без изменения блоки данных, а также участки, которые требуют изменения
  + test - скрипт, генерирующий тестовые данные а также запускающий программу с этим набором данных.
* Следующие модули небыли реализованы:
  + config
  + inotify

Причиной отложенной разработки этих модулей служит то, что их наличие или отсутствие не повлияет на результаты проведения тестирования созданного решения.

## 2.6.Тестирование разработанного решения

Во время тестирования модуля было исследовано время выполнения программы. При этом менялись небольшие блоки информации в различных частях документа (от 1 до 10 блоков, их местоположение в документе определялось случайным образом). Размер вставленного блока был постоянен и равен 50 байтам (50 символов) - его объем практически не влияет на работу алгоритма. Объем исходного файла менялся в пределах от 1 до 21 Мб.

В ходе тестирования были получены следующие данные:

|  |  |  |
| --- | --- | --- |
| размер исходного файла (Мб) | количество изменений (Шт.) | время выполнения (Сек.) |
| 1 | 1 | 8.61 |
| 1 | 2 | 41.35 |
| 1 | 3 | 90.65 |
| 1 | 4 | 107.66 |
| 1 | 5 | 167.3 |
| 1 | 6 | 154.05 |
| 1 | 7 | 115.03 |
| 1 | 8 | 167.15 |
| 1 | 9 | 170.02 |
| 1 | 10 | 214.47 |
| 2 | 1 | 62.34 |
| 2 | 2 | 115.54 |
| 2 | 3 | 163.77 |
| 2 | 4 | 206.21 |
| 2 | 5 | 217.7 |
| 2 | 6 | 296.82 |
| 2 | 7 | 462.35 |
| 2 | 8 | 386.19 |
| 2 | 9 | 470.65 |

|  |  |  |
| --- | --- | --- |
| 2 | 10 | 326.22 |
| 3 | 1 | 131.66 |
| 3 | 2 | 242.88 |
| 3 | 3 | 264.28 |
| 3 | 4 | 475.01 |
| 3 | 5 | 532.37 |
| 8 | 1 | 82.53 |
| 8 | 2 | 524.05 |
| 8 | 3 | 701.81 |
| 13 | 1 | 409.34 |
| 13 | 2 | 1121.28 |
| 13 | 3 | 1731.79 |
| 21 | 1 | 912.31 |
| 21 | 2 | 1827.16 |
| 21 | 3 | 2701.87 |

Таблица 1. Результаты тестирования разработанного модуля

С графиками, основанными на этих данных, можно ознакомиться в Приложении 1.

Разработанный модуль показал себя стабильно работающим с различными объемами данных. Однако время выполнения растет достаточно быстро, что осложняет его тестирование на исходных файлах большого объема.

# Глава 3. Результаты

В данной главе будут подведены итоги о проделанной работе, сделаны выводы и произведена оценка созданного решения.

Кроме того, будут рассмотрены возможные варианты дальнейшего развития созданного модуля.

## 3.1. Дальнейшее развития модуля

* В первую очередь необходимо разработать оставшиеся подмодули системы. После этого её использование станет возможным для резервного копирования данных.
* Далее, учитывая, что целью было разработать прослойку для любого существующего сервиса, необходимо сделать её кроссплатформенной.
* Также необходимо подробнее ознакомиться с исходным кодом различных облачных сервисов и убедиться, что интерфейса, предоставляемого разработанным модулем, достаточно для их корректного взаимодействия.
* В случае если разработанного интерфейса недостаточно - разработать необходимые методы.
* Определить все “узкие горлышки” в программе, оптимизировать использующиеся в этих участках методы (в частности, использовать многопоточность при нахождении измененной части файла).
* Произвести рефакторинг когда (перейти к одному стилю именования переменных и унифицировать используемые технологии)
* Рассмотреть возможность самостоятельной реализации сторонних модулей для увеличения производительности ( в частности, модуля md5 )  
  3.2. Выводы

В ходе работы было проведено тестирование разработанного модуля, которое показало что рост времени, затраченного на определение разницы файлов - самой сложной для вычисления из всех операций, осуществляемый модулем - пропорционально объему файла и количеству изменений, произведенных в файле.

В то же время местоположение изменений практически не влияет на скорость вычислений и общий вид графиков.

Стоит отметить, что некоторые подмодули разработанной системы, возможно, написаны не самым оптимальным образом и нуждаются в более детальном анализе.

В данном случае нам приходится выбирать между объемом трафика, которым придется обмениваться, и временем вычисления данных, необходимых для синхронизации. На данный момент разработанное решение однозначно выигрывает, если экономия трафика более приоритетна, а также в случае высокой производительности компьютеров, на которых размещается описанное приложения. В случаях же, когда скорость обмена приоритетнее и качество Интернет-соединения достаточно высокое - в осуществляемых вычислениях нет необходимости.

## Заключение

В рамах проекта по обеспечению надежности хранения данных при использовании клиент-серверных приложений, в частности облачных хранилищ, был проведен анализ требований и поиск оптимального решения. Разработка собственного модуля в данном случае не является наиболее выгодным решением, поскольку без его внедрения в коммерческие продукты написанное решение не решает проблем, которые были выявлены при анализе рассматриваемой области, в частности доступности использования со стороны пользователей.

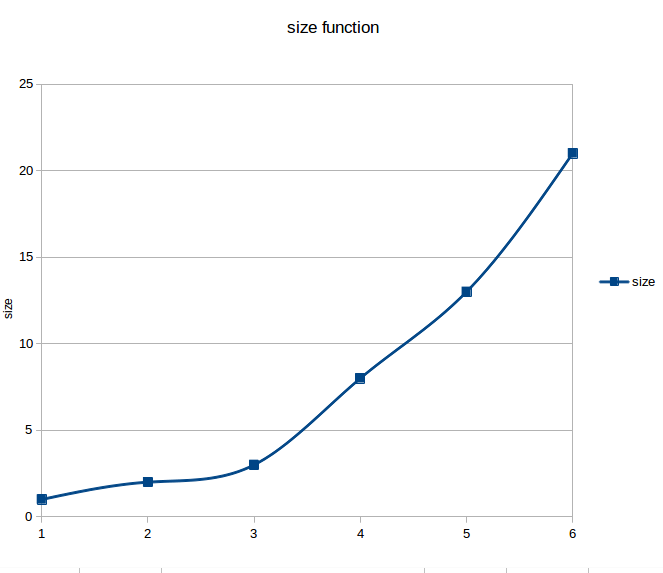
Однако, при дальнейшей разработке модуля и оптимизации алгоритмов для наибольшей скорости работы есть вероятность, что открытость исходного кода спровоцирует некоторые компании, предоставляющие услуги облачного хранения, использовать созданное решение в своих продуктах. Это поможет обеспечить надежное хранение данных в облаке с защитой от произвольного доступа в случае утечки данных с серверов, на которых осуществляется хранение.

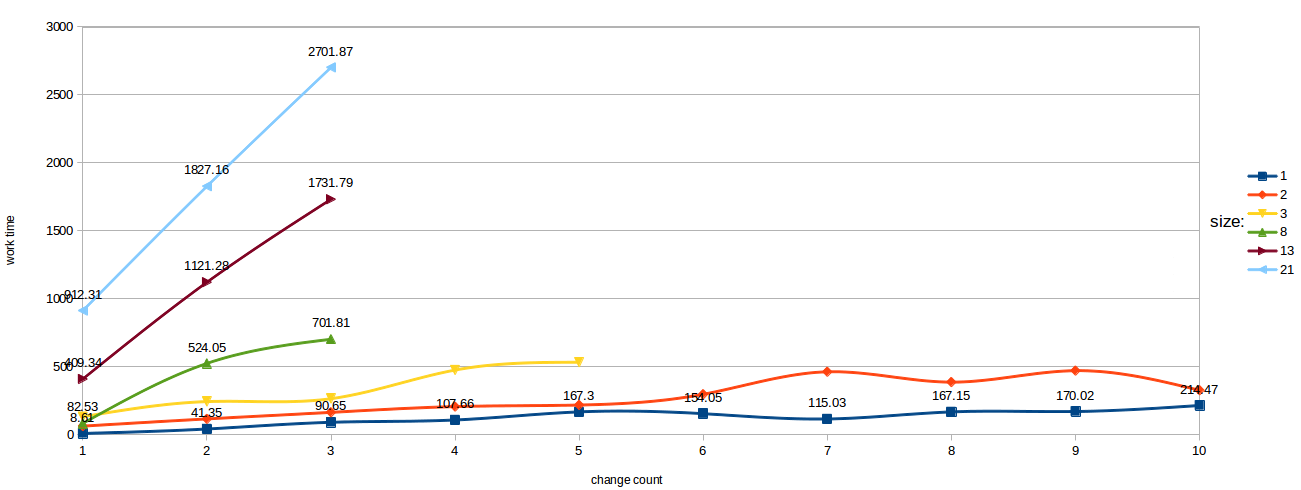
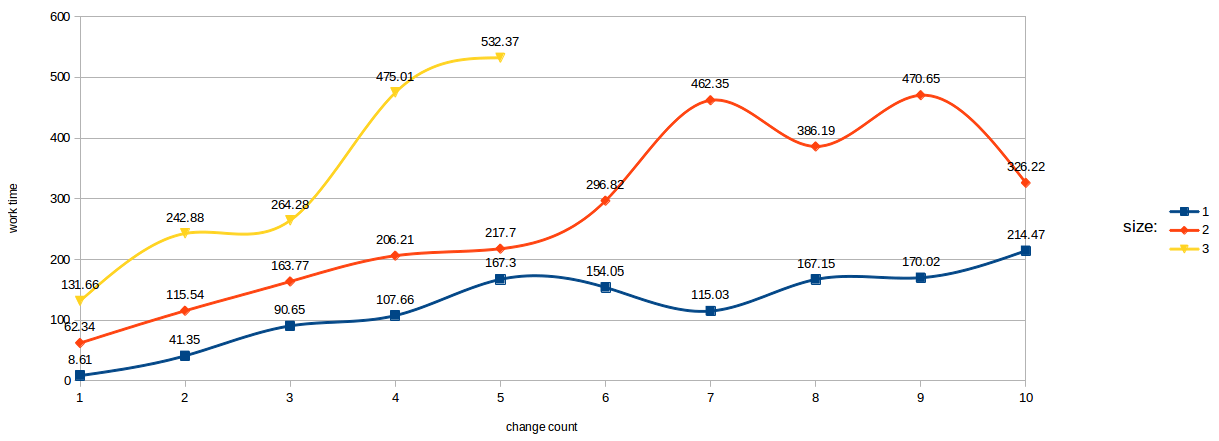
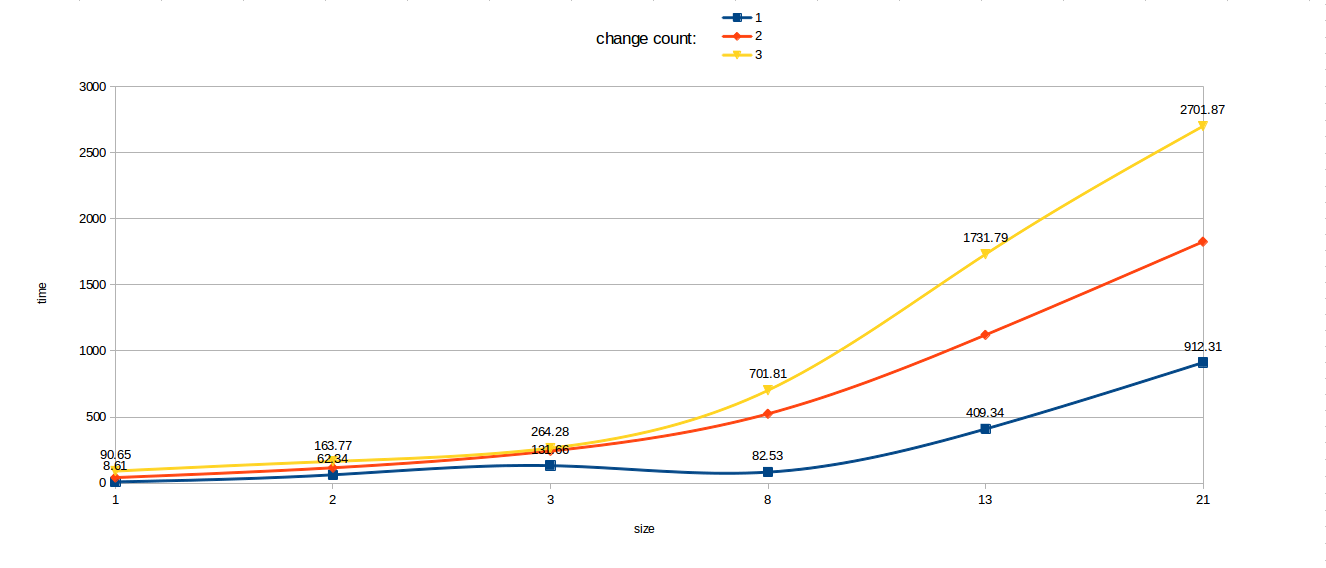
# Список литературы и источников

1. Развитие облачных технологий - [http://www.iaas.su/article/razvitie-oblachnyh-tehnologiy](http://www.iaas.su/article/razvitie-oblachnyh-tehnologiy/)
2. Облачная эластичность - <http://habrahabr.ru/company/abbyy/blog/152947/>
3. Выбор облачного хранилища - <http://romweb.ru/web-servisy/chto-takoe-oblachnoe-hranilishhe-dannyh/>
4. Утечки данных из облачных хранилищ - <http://romweb.ru/web-servisy/kak-oblachnye-hranilishha-sami-ispolzujut-moi-fajly-zashhita-dannyh-oblakah/>
5. Примеры лицензионных соглашений - <http://habrahabr.ru/post/143111/>
6. Лицензионное соглашение mail.ru - <http://habrahabr.ru/post/207780/>
7. Хранение данных в Dropbox - <http://habrahabr.ru/post/119348/>
8. Waula - <http://webtun.com/webservices/2783-obzor-zaschischennogo-oblachnogo-servisa-wuala.html>
9. SpiderOak - http://ru.wikipedia.org/wiki/SpiderOak
10. UbuntuOne - <https://one.ubuntu.com/about/>
11. Закрытие UbuntuOne - <http://blog.canonical.com/2014/04/02/shutting-down-ubuntu-one-file-services/>
12. UbuntuOne Client исходные коды - <https://launchpad.net/ubuntuone-client>
13. Решения пользователей по защите данных - <http://romweb.ru/web-servisy/kak-zashhitit-moi-fajly-v-oblake-realno-dejstvuyushhij-variant/>
14. Boxcryptor - <https://www.boxcryptor.com/ru/boxcryptor>
15. Сервисы, поддерживающиеся Boxcryptor - <https://www.boxcryptor.com/ru/provider>
16. Viivo - <http://www.viivo.com/products/>
17. Cloudfogger - [http://thehackernews.com/2014/01/how-to-encrypt-your-files-before.html](http://thehackernews.com/2014/01/how-to-encrypt-your-files-before.html4)
18. Crypto++ - <http://www.cryptopp.com/docs/ref/>

# Приложение 1. Графики на основе полученных в ходе тестирования данных.

* график изменения размера исходного файла



* график зависимости времени вычисления разницы от количества изменений в файле для исходных файлов разного объема
* график зависимости времени вычисления разницы от количества изменений в файле для исходных файлов разного объема (детальный)
* график зависимости времени времени вычисления от объема, при различных количествах изменений файла

# Приложение 2. Исходный код заголовочных файлов реализованных модулей

* ***filemanager.h***

#ifndef FILEMANAGER\_H

#define FILEMANAGER\_H

#include <boost/filesystem.hpp>

#include <string>

#include <fstream>

#include <sstream>

#include "scrambler.h"

#include "md5.h"

#include "metaworker.h"

#include "updater.h"

class Filemanager {

public:

Filemanager();

~Filemanager();

bool segmentate(std::string file\_path);

bool segmentate(std::string file\_path, long begin, long end);

bool merge(std::string file\_path);

bool update(std::string file\_path);

private:

int buffer\_size;

std::string input\_dir;

std::string output\_dir;

Scrambler \*scrambler;

bool rm\_file(std::string path);

Metaworker \*mworker;

};

#endif

* ***metaworker.h***

#ifndef METAWORKER\_H

#define METAWORKER\_H

#include <string>

#include <iostream>

#include <fstream>

#include <vector>

#include <stdlib.h>

#include <algorithm>

#include "md5.h"

struct metarow {

long position;

std::string clean\_hash;

std::string cipher\_hash;

int start;

int finish;

std::string as\_string() const;

void show() const;

};

class Metaworker {

public:

Metaworker();

~Metaworker();

void load(std::string file\_path);

const metarow &get(int position) const ;

bool set(std::string clean\_hash, std::string cipher\_hash, int start, int finish, int position, const bool replace);

bool set(std::string clean\_hash, std::string cipher\_hash, int start, int finish);

bool remove(int position);

long mdata\_size() const;

void show() const;

bool save();

std::vector<metarow> mdata;//meta-data

bool sort();

private:

std::string mfpath;

std::string output\_dir;

static bool sort\_func(metarow a, metarow b);

bool get\_content\_from\_metafile();

metarow \*generate\_metarow(int position, std::string clean\_hash, std::string cipher\_hash, int start, int finish);

bool write\_metarow(metarow \*row, const bool replace);

long get\_str\_count();

};

#endif

* ***scrambler.h***

#ifndef SCRAMBLER\_H

#define SCRAMBLER\_H

#include <iostream>

#include <iomanip>

#include <cryptopp/modes.h>

#include <cryptopp/aes.h>

#include <cryptopp/filters.h>

class Scrambler {

public:

Scrambler(std::string private\_key);

char \*encode(const char \*bite\_stream, const int length);

char \*decode(const char \*bite\_stream, const int length);

private:

void set\_private\_key(std::string value);

void set\_private\_key\_length(const int &value);

int sz;

byte \*private\_key;

int private\_key\_length;

byte \*iv;

};

#endif

* ***updater.h***

#ifndef UPDATER\_H

#define UPDATER\_H

#include <string>

#include <stack>

#include <unistd.h>

#include <sys/types.h>

#include <sys/stat.h>

#include <fcntl.h>

#include "metaworker.h"

#include "md5.h"

#include <fstream>

#include <map>

class Updater

{

public:

Updater();

std::pair<std::map<int, int>, std::vector<std::pair<long, long> > >get\_similar(const std::string file\_path);

void show\_similar(std::map<int, int> chunks);

void show\_diff(std::vector<std::pair<long, long> > diff);

private:

std::string output\_dir;

std::string file\_path;

int buffer\_size;

long get\_filesize(std::string file\_path);

long get\_old\_filesize();

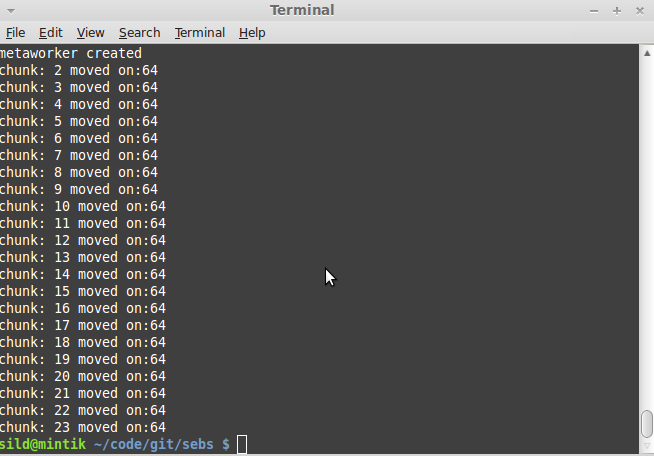
Metaworker \*mworker;

};

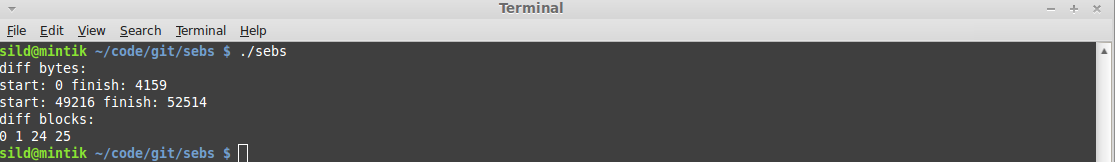
#endif

# Приложение 3. Скриншоты работы программы

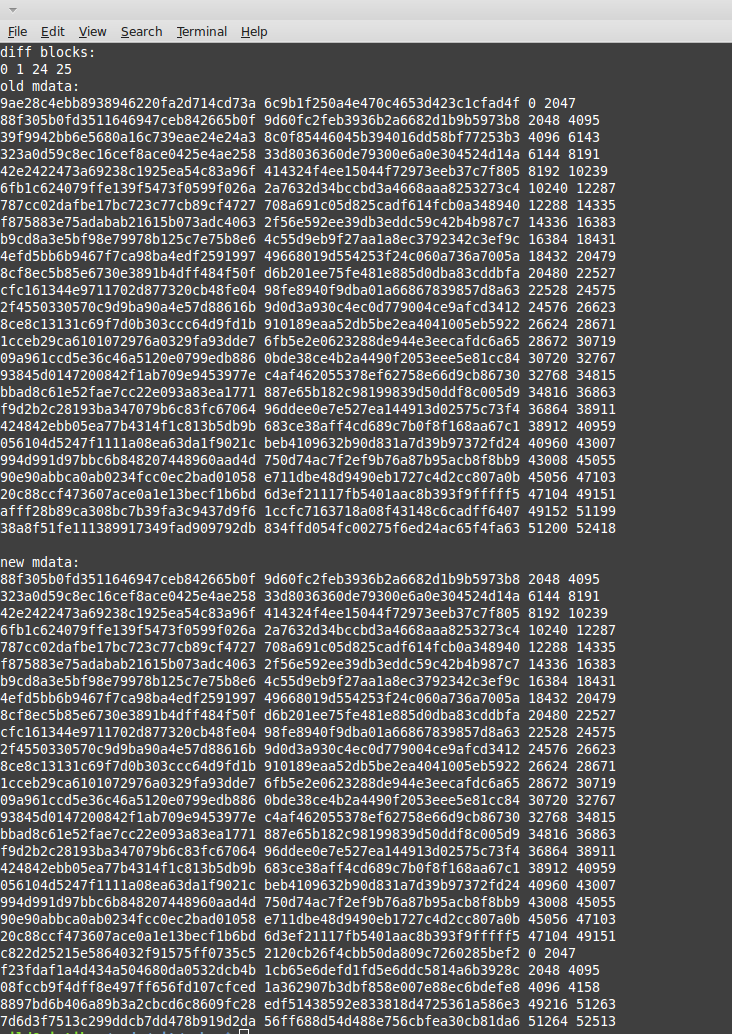
* определение совпавших блоков а также изменения их позиции



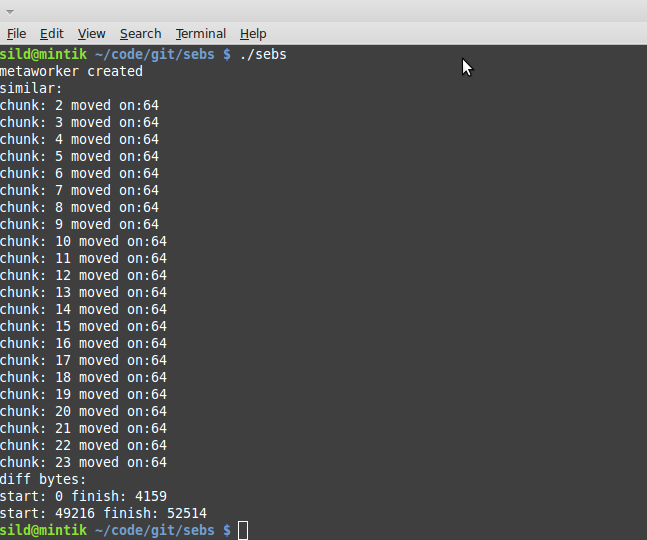
* нахождение отличающихся блоков а также интервалы байт файла, в которых произошли изменения



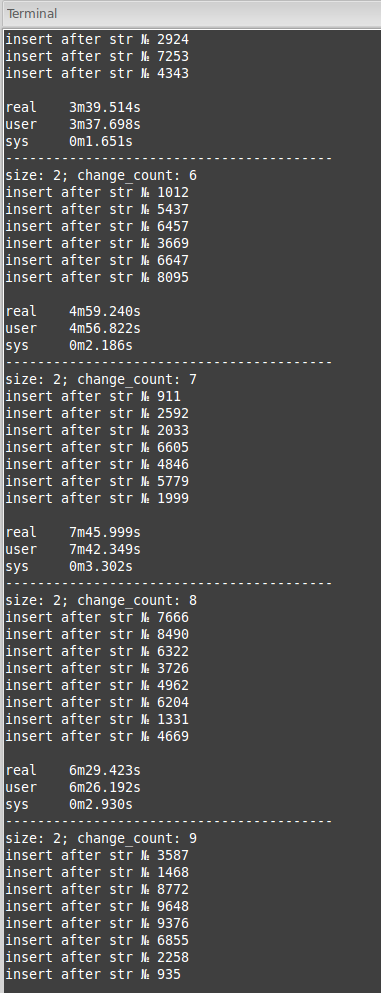
* изменение метаданных



* представление схожих и различных частей файла



* тестирование модуля



# Приложение 4. Рекомендуемые энциклопедические статьи.

1. Интернет - <http://ru.wikipedia.org/wiki/Интернет>
2. Дискета - <http://ru.wikipedia.org/wiki/Дискета>
3. Облачное хранилище данных - <http://ru.wikipedia.org/wiki/Облачное_хранилище_данных>
4. Облачные приложения - <http://ru.wikipedia.org/wiki/SaaS>
5. Dropbox - <http://ru.wikipedia.org/wiki/Dropbox>
6. EncFS - <http://ru.wikipedia.org/wiki/EncFS>
7. Inotify - <http://ru.wikipedia.org/wiki/Inotify>
8. Определение разницы двух файлов - <http://ru.wikipedia.org/wiki/Diff>
9. Наибольшая общая последовательность - <http://ru.wikipedia.org/wiki/Наибольшая_общая_последовательность>
10. Наибольшая общая подстрока - <http://ru.wikipedia.org/wiki/Наибольшая_общая_подстрока>