Федеральное агентство связи
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и информатики»
(СибГУТИ)

Кафедра ПМиК
Допустить к защите
зав. кафедрой: проф., д.т.н.

_____ Фионов А.Н.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

РЕАЛИЗАЦИЯ И ИССЛЕДОВАНИЕ ШИФРОВ ШЕННОНА

Студент Чусовитин Антон Романович / /

Факультет ИВТ Группа ИП-311

Руководитель Ракитский Антон Андреевич / /

Новосибирск _2017_ г.

Оглавление

ВВЕДЕНИЕ 3
1. КРИПТОГРАФИЯ 4
1.1 Криптографический алгоритм 4
1.2 Симметричные криптосистемы 5
1.3 Поточный шифр 6
1.4 Синхронные поточные шифры. 6
2 КРИПТОГРАФИЧЕСКАЯ СТОЙКОСТЬ 9
2.1 Основная информация 9
2.2 Абсолютно стойкая система 10
2.3 Оценка криптостойкости систем шифрования 11
3 ШИФР ВЕРНАМА 12
3.1 Описание шифра 12
3.2 Криптографическая стойкость 13
3.3 Недостатки шифра 13
4 ШИФР ШЕННОНА 15
5 РАЗРАБОТКА ПРОЕКТНОЙ РАБОТЫ 17
5.1 Обоснование выбора языка 17
5.2 Описание реализации шифра 18
5.3 Реализация 18
6 СТАТИСТИЧЕСКИЕ ТЕСТЫ ДЛЯ ПОТОЧНЫХ ШИФРОВ 24
6.1 Обоснование актуальности тестов 24
6.2 Исследование реализации 25
ВЫВОД 29
ЗАКЛЮЧЕНИЕ 30
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 31
Приложение А. 32

ВВЕДЕНИЕ

Передача сообщения между людьми была всегда очень важной задачей. Но при перехвате сообщения его мог узнать третий человек. Для защиты сообщений с давних времен придумывались шифры. Но с развитием технологий, взлом шифров становится всё проще и безопасность сообщения становится под вопросом.

С появлением ЭВМ методы шифрования также стали сложнее. Основной сложностью стала передача закрытого ключа от одного человека к другому. Каждый раз встречаться и передавать лично с закрытым ключом неудобно. Если передавать закрытый ключ по каналу связи, то его может перехватить третье лицо.

Вернам предложил систему, которую невозможно взломать в реальных условиях. Но в системе встречается множество недостатков, из-за которых на практике невозможно использовать шифр для передачи множества сообщений. Потому что если использовать один и тот же закрытый ключ, то третье лицо сможет выявить закономерность и восстановить закрытый ключ.

Шеннон предложил систему, которая поможет решить данные недостатки и позволит открыто передавать необходимые данные для генерации закрытого ключа. Но доказательства стойкости и надежности данной системы отсутствовали. В 2016 году Рябко Б.Я. в своей работе [6] доказал данный метод шифрования.

В данной работе будет разработана реализация клиент-серверного приложения, благодаря которому клиенты смогут обмениваться сообщениями, зашифрованными методом Шеннона. Также будут проведены статистические тесты, для выявления слабых точек реализации и указаны важные аспекты реализации данной криптосистемы.

1. КРИПТОГРАФИЯ

1.1 Криптографический алгоритм

Криптография — наука, занимающаяся методами преобразования информации, не позволяющим противнику извлечь данные из перехватываемых им сообщений. Криптографический алгоритм — система преобразований входной информации в шифротекст с целью обеспечить секретность передаваемых данных. Неотъемлемой частью и главной особенностью любого алгоритма шифрования является использование ключа шифрования, при помощи которого осуществляется конкретное преобразование из совокупности возможных вариантов для данного шифра. [1] Основная идея шифрования заключается в том, что злоумышленник, которому удалось перехватить передаваемую информацию, имея зашифрованные данные и не имея к ним ключа, не будет иметь возможности ни прочесть, ни изменить перехваченную информацию.

Всё разнообразие существующих на настоящий момент шифров можно разделить по двум крупным классификациям. Первая классификация разделяет алгоритмы шифрования на две категории в зависимости от структуры используемых ключей:

1) Симметричный метод, использующий один и тот же ключ, как для шифрования, так и для дешифрования
2) Асимметричный метод, использующий для тех же целей два различных ключа.

В зависимости от объема шифруемых данных, а также способа их обработки, шифры подразделяются на:

1) Блоковый шифр, шифрующий блоки информации фиксированной длины, блок информации обрабатывается целиком
2) Поточный шифр, шифрующий информацию по мере поступления, обрабатывает биты открытого и закрытого текста в режиме, приближенном к реальному времени.

В данной работе были реализованы исследования над двумя алгоритмами шифрования данных, которые относятся к поточным симметричным шифрам, поэтому особое внимание уделяется именно этой категории криптографических алгоритмов.

1.2 Симметричные криптосистемы

В симметричных криптосистемах шифрование и расшифровывание исходного сообщения происходит одним и тем же ключом. Любой, кто имеет доступ к ключу, может расшифровать исходное сообщение. Для такой системы все ключи должны держаться в секрете. Отсюда следует что ключ должен быть доступен только тем, кто должен получить сообщение. Схема симметричной криптосистемы шифрования изображена на рисунке 1.1:

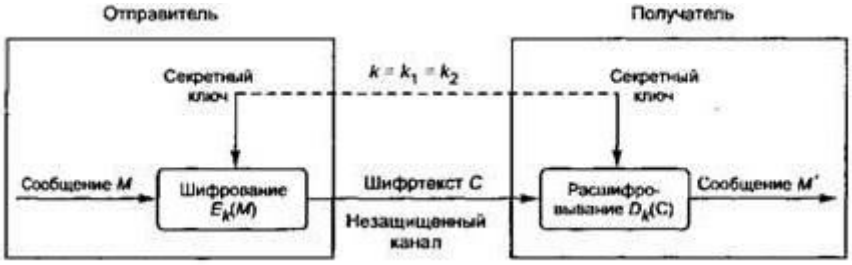

Рисунок 1.1 Схема симметричной системы шифрования.

Данную криптосистему можно охарактеризовать, как систему с высокой скоростью шифрования, обеспечивающую секретность, подлинность и целостность исходного сообщения. Секретность передачи данных зависит от надежности шифра и от того, насколько хорошо хранится закрытый ключ.

Закрытый ключ представляет из себя файл либо массив с данными, которые хранятся на персональном носителе, что позволяет иметь доступ к закрытому ключу только владельцу. Подлинность передаваемого сообщения достигается благодаря тому, что невозможно модифицировать передаваемое сообщение без знания закрытого ключа для шифрования. Модифицированное сообщение не может быть зашифровано без закрытого ключа.

Целостность передаваемых данных обеспечивается благодаря вставке контрольной суммы исходного сообщения. Получатель после расшифровывания сможет проверить целостность сообщения.

При подготовке к шифрованию необходимо передать закрытый ключ между отправителем и получателем. От надежности передачи закрытого ключа, зависит и безопасность передачи шифротекста. На каждую пару пользователей должна приходиться уникальная пара ключей.

1.3 Поточный шифр

Поточный шифр — это симметричный шифр, в котором каждый символ текста преобразуется в символ шифрованного текста в зависимости не только от используемого ключа, но и от его расположения в потоке открытого текста. [6] Простейший пример поточного шифра:

1) Генерируется гамма-последовательность K.
2) Исходный текст M.
3) К гамма-последовательности и исходному тексту применяется битовая операция XOR и получается шифротекста C.
4) Расшифровка происходит операцией XOR между гамма-последовательностью и шифротекстом получая исходный текст M.

Отсюда следует, что если гамма-последовательность не имеет периода и выбирается случайно, то взломать шифр невозможно. [6]
Но поточный шифр имеет и свои минусы – из-за того, что закрытые ключи одинаковой длины с исходным сообщением в реальных системах не встречаются, на практике применяются ключи меньшей длины. С помощью него генерируется псевдослучайная гаммирующая последовательность. В будущем такое свойство гамма-последовательности, как псевдослучайность, может быть использовано при совершении атаки на шифр.

Если при генерации гамма-последовательности произойдет искажение одного любого бита, то при передаче по каналу связи и расшифровывании будет искажен всего лишь один бит, а остальная часть будет расшифрована правильно. Но если произойдет потеря 1 бита при передаче по каналу связи, то расшифровать сообщение дальше потерянного бита будет невозможно.

Практически во всех каналах передачи данных присутствуют помехи. Во избежание потери информации решают проблему синхронизации шифрования и расшифрования текста – применяют синхронные поточные шифры.

1.4 Синхронные поточные шифры.

Синхронные поточные шифры – такие шифры в которых поток ключей генерируется независимо от исходного текста и шифротекста. [6]

Во время шифрования гамма-последовательность выдает биты для

шифрования исходного сообщения, которые одинаковы с битами для расшифровывания. Потеря знака шифротекста приведет к нарушению синхронизации между двумя гамма-последовательностями и невозможность расшифровать сообщение. В этой ситуации отправитель и получатель должны синхронизироваться. Синхронизация происходит вставкой в передаваемый шифротекста специальных маркировок. Благодаря этому не правильное расшифровывание передаваемого шифротекста происходит лишь до того момента, пока на стороне получателя не будет принят один из маркеров. Для выполнения синхронизации ни одна часть закрытого ключа не должна повторяться.

Так как генерируется один и тот же поток ключей, как для шифрования, так и для дешифрования, следовательно, вывод ключевых элементов должен быть предопределен. Если данная система реализуется на конечном автомате, то рано или поздно последовательность придет к повторению, такая генерация потока называется периодической. Все генераторы ключевого потока являются периодическими, за исключением одноразовых блокнотов.

Плюсы синхронного поточного шифра:

1) Отсутствие распространения ошибок – только искаженные биты будут расшифрованы неправильно, все предшествующие и последующие биты не будут подвержены изменениям.
2) Защищает от подмены шифротекста данный факт будет заметен, так как действие подмены приведет к потере синхронизации.

Минусы синхронного поточного шифра:

1) Если злоумышленник знает исходный текст, то он сможет подменой битов изменить данные при расшифровке шифротекста.

Однако, синхронные потоковые шифры имею уязвимость к атакам на основе изменения отдельных бит данных шифротекста. Злоумышленник может изменить эти отдельные биты так, что шифротекста расшифруется так, как ему это будет выгодно.

Согласно Райнеру Рюппелю можно выделить четыре основных подхода к проектированию поточных шифров [6]:

1) Системно-теоретический подход - основан на создании для криптоаналитика сложной, ранее неисследованной проблемы.

2) Сложно-теоретический подход - основан на сложной, но известной проблеме (например, факторизация чисел или дискретное логарифмирование).
3) Информационно-технический подход - основан на попытке утаить открытый текст от криптоаналитика – вне зависимости от того сколько времени потрачено на дешифрование, криптоаналитик не найдёт однозначного решения.
4) Рандомизированный подход - основан на создании объёмной задачи; криптограф тем самым пытается сделать решение задачи расшифрования физически невозможной. Например, криптограф может зашифровать некоторую статью, а ключом будут указания на то, какие части статьи были использованы при шифровании. Криптоаналитику придётся перебрать все случайные комбинации частей статьи, прежде чем ему повезёт, и он определит ключ.

Если атакующий обладает неограниченными вычислительными ресурсами и временем, то единственной реализуемой криптосистемой, защищенной от такого воздействия является одноразовый блокнот.

2 КРИПТОГРАФИЧЕСКАЯ СТОЙКОСТЬ

2.1 Основная информация

Криптографическая стойкость — одно из центральных понятий криптографии и наиболее значимый параметр, характеризующий алгоритмы шифрования. Стойкостью шифра называют его способность противостоять возможным направленным на него атакам. Криптостойкость зашифрованной информации напрямую зависит от возможности попыток несанкционированного чтения данных.

Криптоанализ — наука, позволяющая найти относительную стойкость алгоритма, под данным понятием также понимается попытка найти уязвимость в алгоритме, выяснить ключ или вскрыть шифр. Абсолютно стойким алгоритмом считается такой алгоритм, на который от атакующего потребуются недостижимые на практике вычислительные ресурсы либо колоссальное количество времени для расшифровки перехваченного сообщения, такое, что по его истечению сообщение потеряет ценность. Часто можно только доказать уязвимость только алгоритма.

К основным видам атакам, направленных на вскрытие поточных шифров, можно отнести такие, как:

1) Силовые – основанные на полном переборе всех возможных вариантов. Сложность этого вида атаки зависит от таких параметров, как размер ключей, открытого текста и т.д.
2) Статистические – направленные на изучение выходной последовательности данных, статистических свойств шифрующей системы. А также метод анализа сложности последовательности – способ направлен на поиск метода генерировать идентичную гамма-последовательность, но более простым методом.
3) Аналитические – методы применяются при известном описании генератора, открытого и закрытого текста. Целью данной атаки является определить начальный ключ.

В зависимости от криптостойкости алгоритмов они делятся на абсолютно стойкие и достаточно стойкие системы. Шифры, исследуемые в данной работе, относятся к категории абсолютно стойких систем, поэтому более подробно рассмотрим свойства алгоритмов этой категории.

2.2 Абсолютно стойкая система

Абсолютно стойкой или обладающей теоретической стойкостью называется такая система, уязвимость которой невозможно доказать ни теоретически, ни практически, даже если у атакующего имеются бесконечно большие вычислительные ресурсы.

Доказательство существования данных систем шифрования, а также необходимые и достаточные условия для совершенной секретности, были определены Клодом Шенноном в работе “Теория связи в секретных системах”. [2]

Приведенные ниже параметры Шеннон определил, как необходимые для создания абсолютно стойких систем:

1) Закрытый ключ генерируется уникально для каждого передаваемого сообщения.
2) Вероятность появления каждого нового символа закрытого ключа равна и символы последовательно независимы и случайны.
3) Длина передаваемого сообщения и длина закрытого ключа должны совпадать или закрытый ключ должен быть длиннее передаваемого сообщения.
4) Передаваемое сообщение имеет некоторую избыточность, для того чтобы оценить правильность расшифровки шифротекста.

А также обозначил недостаток, имеющийся у совершенно секретных систем, который проявляется в том, что для передачи большого объема данных требуется посылать ключ шифрования эквивалентного с сообщением объема.

Шеннон доказал, что примером абсолютно стойкой системы является шифр Вернама, иначе называемый одноразовый блокнот. Верное использование данного шифра не позволит получить злоумышленнику какую-либо информацию из зашифрованной информации из-за того, что любой бит перехваченного сообщения можно лишь угадать с вероятностью 1/2.

На данный момент практическое применение криптосистем, удовлетворяющих требованиям абсолютной стойкости, ограничено из соображений стоимости и удобства пользования.

2.3 Оценка криптостойкости систем шифрования

Все современные шифры построены на принципах Кирхгофа, которые говорят о том, что секретность шифра обеспечивается не секретностью алгоритма, а секретностью ключа. [3] Анализ надежности криптосистемы всегда должен производиться исходя из того, что злоумышленник обладает всей возможной информацией о применяемом криптоалгоритме и неизвестен ему только ключ шифрования. Это свойства оценки криптостойкости имеет место быть по причине того, что даже если держать весь алгоритм шифрования в тайне, информация рано или поздно может быть скомпрометирована.

Стойкость криптосистемы зависит от таких параметров, как сложность алгоритма, длина ключа, методы реализации шифра и т.д. Процесс оценки криптостойкости систем условно можно разделить на три этапа:

1) Оценить сколько времени и вычислительных мощностей потребуется для полного перебора всех ключей. Сложность вычислительных алгоритмов можно оценивать числом элементарных операций, с учетом затраты на их выполнение. Это число должно выходить за пределы возможностей современных компьютерных систем.

2) Поиск уязвимостей алгоритма шифрования к известным атакам, таким как: линейный криптоанализ, дифференциальный криптоанализ и другие. Также, оценка статистической безопасности шифра, о которой говорит отсутствие зависимости между входной и выходной последовательностями и между ключами, используемыми в процессе шифрования, а также широко проявляемое диффузионное свойство. В этот этап входит проверка атаки на реализацию, которая происходит на конкретный программный-комплекс.

3) Длительность изучения алгоритма и реализаций на программных-комплексах, так как чем дольше происходит анализ, тем больше доказывается криптостойкость алгоритма.

Шифр считается достаточно надежным, если отсутствует возможность раскрыть его способом более эффективным, чем полным перебором по всему ключевому пространству.

3 ШИФР ВЕРНАМА

3.1 Описание шифра

Шифр Вернама, изобретенный в 1917 году, является симметричной системой шифрования, а также разновидностью криптосистемы одноразовых блокнотов. Данная криптосистема была предложена для использования в системе телеграфных сообщений. Текст сообщения имел бинарный вид, а секретный ключ представлялся как случайный набор букв такого же алфавита, что и сообщение.

Для того, чтобы получить шифротекст, открытый текст объединяется с секретным ключом при помощи операции исключающее ИЛИ. К примеру, для шифрования двоичной последовательности (100110) с применением закрытого ключа (010010) будет получен шифротекст (110100). Для того, чтобы получить исходное сообщения (100110) из полученного шифротекста достаточно применить обратную операцию исключающее ИЛИ к принятому шифротексту (110100) и закрытому ключу (010010).

Для того, чтобы шифр имел полную криптографическую стойкость, закрытый ключ должен обладать как минимум тремя, приведенными ниже, свойствами:

1) Иметь случайное равномерное и равновероятное распределение двоичных символов.
2) Размер закрытого ключа должен полностью совпадать с размером заданного сообщения.
3) Закрытый ключ должен использоваться ровно один раз на сообщение.

Важной особенностью данного криптоалгоритма является то, что при применении шифра Вернама за перехваченным шифрованным текстом может скрываться абсолютно любой открытый текст. Аналогично, совершенно любому открытому тексту также можно подобрать гамму, которая будет порождать данный шифротекст.

Поскольку гамма является ключом, то при перехвате шифрованного сообщения невозможно отвергнуть ни одного открытого текста, имеющего такую же длину. [4]

3.2 Криптографическая стойкость

Клод Шеннон в 1945 году доказал абсолютную стойкость шифра Вернама. При перехвате шифртекста третьим лицом, перехватчик не сможет узнать никакой информации об исходном сообщении или закрытом ключе. В криптографии шифр Вернама считается идеально безопасной системой. Для реализации такой системы необходимо обеспечить уникальное гаммирование исходного сообщения. Важным условием гаммирования являются:

1) При шифровании нового сообщения используется новая гамма-последовательность. Повторное использование гамма-последовательности невозможно из-за свойств операции “исключающее ИЛИ”. Рассмотрим пример:

   $m'_1 = m_1 \oplus c$ (3.1)

   $m'_2 = m_2 \oplus c$ (3.2)

   $m'_1 \oplus m'_2 = (m_1 \oplus c) \oplus (m_2 \oplus c) = m_1 \oplus m_2$ (3.3)

   Результат не зависит от гамма-последовательности С и полностью зависит от исходного сообщения. Так как естественные языки имеют высокую избыточность, то результат поддается частотному анализу, соответственно можно подобрать исходные тексты и не важна гамма-последовательность.
2) Если гамма-последовательность будет сгенерирована на аппаратных генераторах случайных чисел, то гамма не будет считаться случайной и с помощью перебора, возможно подобрать начальное состояние генератора.
3) Длина гамма-последовательности не должна быть короче длины исходного сообщения, так как при выборе гамма-последовательности меньшей длины можно будет подобрать размер, проанализировать блоки шифротекста и подобрать биты гамма-последовательности.

3.3 Недостатки шифра

Несмотря на то, что шифр Вернама является абсолютно криптографически стойкой и самой безопасной системой, она имеет ряд существенных недостатков.

1) Главным необходимым условием криптографической стойкости шифра является случайность последовательности закрытого ключа. Полученная при помощи любого алгоритма последовательность

считается не абсолютно случайной, а псевдослучайной.

2) Невозможность подтверждения целостности и подлинности переданного сообщения получателю. Получатель не может удостовериться, что сообщение пришло без искажений, соответствует исходному, а также подлинность отправителя, так как при перехвате и расшифровке шифротекста третьим лицом, он может заменить передаваемое сообщение своим сообщением точно такой же длины и зашифровать взломанным закрытым ключом.
3) Необходимость иметь для множества сообщений множество закрытых уникальных ключей. Это может привести к разрыву канала между двумя лицами до момента получения новых сгенерированных закрытых ключей. Также невозможно заранее знать размер передаваемого сообщения.
4) Закрытые ключи нельзя передавать по каналу связи, в виду его недостаточной защищенности – шифр будет защищенным ровно настолько, насколько защищен канал связи. При этом, принимая в учет тот факт, что ключ имеет ту же длину, что и сообщение, передать ключ по защищенному каналу не будет хоть немного проще, чем передать по каналу само сообщение.
5) Применение повторного ключа влечет за собой подбор закрытого ключа и последующее вскрытие шифра.

4 ШИФР ШЕННОНА

Шифры с бегущим ключом - это такие шифры, в которых исходное сообщение, ключи и зашифрованное сообщение представлены последовательностями символов одного языка.

Шифр с бегущим ключом является системой типа Вернама, в которой ключ является не случайной последовательностью, а осмысленным текстом. В своей работе «Теория связи в секретных системах», Шеннон говорил о значительных минусах данного метода шифрования. Одним из наиболее существенных являлось замечание, касающееся избыточности используемого в процессе шифрования текста. Описываемый недостаток приводил к легкому обнаружению статистических связей, как следствие отсутствие высокой надежности шифрования.

Также, в описываемой работе, Шеннон предложил элементарное в понятиях выполнимости усовершенствование данного шифра таким образом, что система не может быть решена в отсутствии ключевого элемента. Было предложено два варианта реализации улучшенной системы с бегущим ключом, которые являлись простыми в реализации, но в тоже время достаточно надежными.

Идея первого метода шифрования заключалась в том, что вместо единственного текста на английском языке использовалось D различных текстов, которые прибавлялись к исходному сообщению посредством операции сложения по модулю 2. С увеличением количества ключей надежность шифра будет возрастать.

Суть второго метода шифрования заключается в использовании только N-ых букв текста, все остальные промежуточные символы опускаются и далее в процессе шифрования не используются. Благодаря такому способу выборки символов текста, выбор каждой следующей буквы будет практически независим. Этот метод также обеспечивает достаточно высокую надежность шифра.

Шеннон описал данные шифры, но не исследовал их свойства и не привел никаких доказательств стойкости и надежности приведенных методов шифрования.

В 2016 году Рябко Б.Я и Ракитский А.А в своей работе [7] доказали оба шифра. Более того первый шифр Шеннона не может быть дешифрован без ключа, если $d \geq 4$ различных текстов складываются с сообщением (т.е. используются в качестве ключа). В случае второго шифра

дешифровать сообщение невозможно без наличия ключа. В работе [7] подробно приведены доказательства данных шифров.

В моей работе был реализован первый из приведенных Шенноном алгоритм шифрования, а также исследована его стойкость и статистическая зависимость выходных последовательностей.

5 РАЗРАБОТКА ПРОЕКТНОЙ РАБОТЫ

5.1 Обоснование выбора языка

Качество программного обеспечения. Основным преимуществом языка Python является удобочитаемость и высокое качество кода. Код читается намного легче, что ведет к многоразовому использованию и удобному дальнейшему обслуживанию. Единообразие оформления кода упрощает работу тех, кто не участвовал в проекте.

По сравнению с такими языками как C, C++, Java и C, Python повышает производительность разработчика. Объем кода написанного на языке Python в основном составляет 1/3 от такого же кода на языке C. Тем самым разработчик тратит меньше времени на отладку. Также программы на языке Python запускаются сразу, не проходя через этапы компиляции и связывания.

Большая часть кода на языке Python кроссплатформенная. Перенос программы из операционной системы Windows в Linux не требует от разработчика переписывания исходного кода. Python также предоставляет возможность создания переносимых интерфейсов, доступа к базам данных, веб-интерфейсов и многих других программ.

Стандартные библиотеки Python предоставляют возможности работы со многими СУБД, выполнять парсинг данных, работа со множеством сетевых протоколов. Сообщество языка очень активно развивает множество библиотек для работ с математическими вычислениями, доступам к последовательному порту, разработка игровых движков, веб-программирование.

В языке отлично реализована интеграция с другими языками. Например, в коде на языке Python мы можем вызывать стандартные функции библиотек C, C++ и так же вызываться в программах C, C++.

Python максимально оптимизирован для достижения высокой скорости разработки, у языка простой и понятный синтаксис, динамическая типизация, отсутствует этап компиляции.

Как вывод язык увеличивает производительность разработчика во много раз по сравнению с другими языками.

5.2 Описание реализации шифра

Чтобы создать защищенный канал общения между двумя пользователями необходимо снабдить клиентов файлами ключей. Для соединения двух клиентов случайным образом выбирается набор из 128 файлов с ключами, которые передаются обоим клиентам.

После этого клиенты высчитывают по алгоритму Диффи-Хеллмана свой закрытый ключ. Размерность закрытого ключа обязательно должна составлять 128 бит. В полученной последовательности закрытого ключа каждый бит данных обозначает порядковый номер файла. Если бит последовательности закрытого ключа равен единице, то файл с ключом будет использован при шифровании, в ином случае – файл не выбирается.

Первый клиент приступает к этапу шифрования сообщения, используя выбранные при помощи закрытого ключа файлы. Процесс шифрования построен таким образом, что основной этап работы сводится к выполнению операции «исключающее ИЛИ» между исходным сообщением и набором выбранных файлов. Затем, после завершения шифрования сообщения, первый клиент отправляет полученную зашифрованную последовательность второму клиенту.

Второй клиент принимает зашифрованную последовательность и приступает к расшифровке информации. В процессе расшифровки он использует файлы, полученные при помощи закрытого ключа, а значит файлы аналогичные тем, что использовал первый пользователь при шифровании. Выбрав необходимые ему файлы, второй клиент производит обратную операцию для получения исходного незашифрованного сообщения. Этап расшифрования выполняется таким образом, что основной операцией будет являться операция «исключающее ИЛИ» между исходным сообщением и набором выбранных файлов.

5.3 Реализация

Uml диаграмма проекта представляет из себя схему представленную на рисунке 5.1

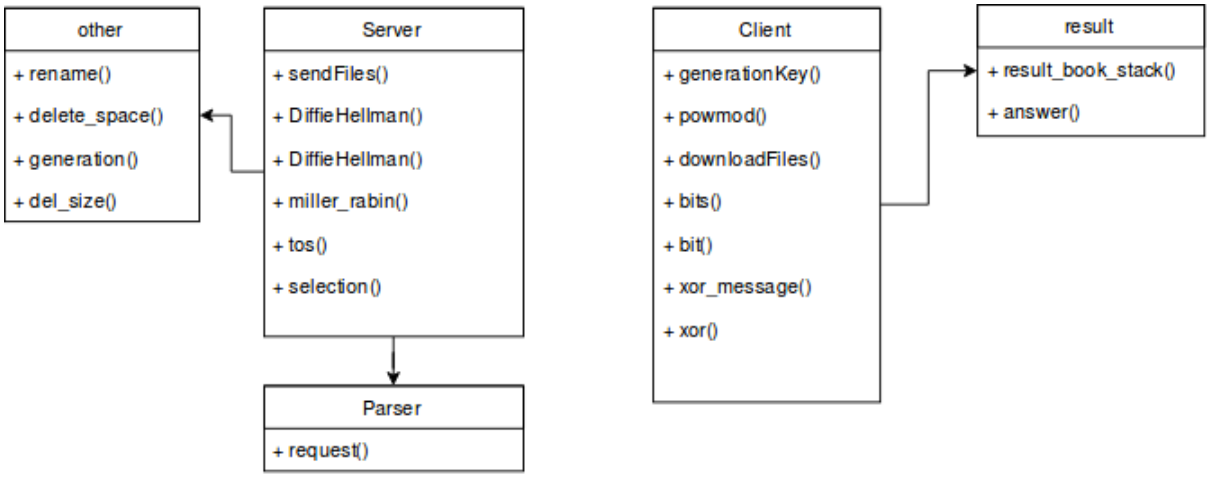

Рисунок 5.1 – UML-диаграмма проекта.

1. Выбор текстов для шифрования сервером

Клиент устанавливает соединение по протоколу TCP с сервером. Сервер в ответ из множества файлов случайным образом выбирает 128 файлов, которые будут переданы клиенту.

Листинг 5.1 Функция выбора текстов для передачи клиенту.

```
def selection(self):
    list = os.listdir("textCode/")
    len_list = len(list)
    answer = []
    while len(ans) != 128:
       select = random.randint(0,lenList-1)
       if (list[select] in answer) == False:
          answer.append(list[select])
    return answer
```

list — Содержит список из всех файлов в директории.
len_list — Содержит количество файлов в директории.
answer — Выходной список размерностью 128 записей.
Select — Номер случайно выбранного документа из списка.

Сервер считывает 262Кб, что представляет достаточное количество для шифрования сообщения клиента, из каждого выбранного файла. Затем, по открытому соединению передает его клиенту. После передачи каждого файла, сервер отправляет системный запрос «next», указывающий

на передачу следующего файла.

Листинг 5.2 Функция передачи выбранных файлов клиенту.

```
def send_files(self):
    list = self.selection()
    in_len = 262144
    for text in list:
       file = open("textCode/" + text, "rb")
       line = file.read(in_len)
       self.request.send(line.encode())
       self.request.send(b"next")
       file.close()
```

list — Результат функции selection. Содержит 128 имен выбранных файлов.

in_len — Количество считываемых с файла данных и передаваемых.

line — Считанные данные из файла

Клиент принимает файлы от сервера, при системной команде «next» начинается запись в следующий файл. После того, как приняты 128 файлов функция завершается.

Листинг 5.3 Функция приема файлов от сервера.

```
def downloadFiles(server):
   data = ""
   for i in range(0, 128):
      file = open("text/"+str(i)+".txt", "wb)
      while True:
         data = server.recv(262144*2)
         if data == b"next“ or not data:
            break
         else:
            file.write(data)
      file.close()
```

data — Строка, в которую будут записаны принятые от сервера данные.

2. Закрытый ключ

Первый клиент высчитывает параметры закрытого ключа по алгоритму Диффи-Хеллмана и передает параметры второму клиенту. Второй клиент в свою очередь высчитывает и отправляет свой открытый ключ для расчета закрытого ключа.

Листинг 5.4 Функция расчета закрытого ключа 1 клиентом.

```
def DiffieHellman(self):
    self.Q = random.randint((2**128), (2**129)-1)
    while self.miller_rabin(self.Q) == False or self.miller_rabin(2
* self.Q + 1) == False:
        self.Q = random.randint((2**128), (2**129)-1)
    self.P = 2 * self.Q + 1
    self.G = random.randint((2**128), (2**129)-1)
    while (self.powmod(self.G, self.Q, self.P) == 1):
        self.G = random.randint((2**128), (2**129)-1)
    self.Xa = random.randint((2**128), (2**129)-1)
    self.Ya = self.powmod(self.G, self.Xa, self.P)
    self.request.send(str(self.P).encode())
    self.request.send(str(self.G).encode())
    self.request.send(str(self.Ya).encode())
    self.Yb = int(self.request.recv(1024).decode())
    self.Zb = self.powmod(self.Yb, self.Xa, self.P)
```

Листинг 5.5 Функция расчета закрытого ключа 2 клиентом.

```
def generation_key(server):
    P = int(server.recv(1024).decode())
    G = int(server.recv(1024).decode())
    Ya = int(server.recv(1024).decode())
    Xb = random.randint((2**128), (2**129)-1)
    Yb = powmod(G, Xb, P)
    server.send(str(Yb).encode())
    return powmod(Ya, Xb, P)
```

3. Выбор текстов для шифрования клиентами

Клиенты проходят по битам закрытого ключа для определения списка необходимых для шифрования текстов. Если бит равен единице, то текст необходим для шифрования, иначе нет.

Листинг 5.6 Функция разбиения числа на биты.

```
def bits(n):
    list_bits = []
    for i in range(0, 128):
        a = bit(n, i)
        list_bits.append(a)
    return list_bits
```

Листинг 3.2 Функция возвращает бит стоящий на позиции num.

```
def bit(num, pos):
    return (num & (1 << pos)) >> pos
```

list_bits — Выходной список, в котором хранятся значение 0 или 1, в зависимости от того, подходит нам файл или нет.

4. Шифрование информации пользователем

Первый клиент вводит сообщение и приступает к его шифрованию. Исходная введенная последовательность шифруется при помощи вспомогательной функции XOR с выбранными файлами.

Листинг 5.7 Функция шифрования/расшифровывания сообщения.

```
def xor_message(text, list):
    for i in range(0, 128):
        if list[i] == 1:
            with open("text/{}.txt".format(i), "rb") as f:
                text = xor(text, bytearray(f.read()))
    return text
```

list — Список из 0 и 1 обозначающие подходит ли файл для шифрования.

Листинг 5.8 Функция xor сообщения с файлом шифрования.

```
def xor(text1, text2):
    lenText2 = len(text2)
    j = 0
    for i in range(len(text1)):
        text1[i] = text1[i] ^ text2[j]
        j += 1
        if j == lenText2-1:
            j = 0
```

```
return text1
```

5. Отправка сообщения

Первый клиент отправляет второму клиенту зашифрованное сообщение.

Листинг 5.1 Функция отправки сообщения клиенту.

```
def send_message(server, message):
    server.send(line.encode())
```

6. Приём сообщения

Второй пользователь принимает сообщение и производит операцию расшифровывания такую же как в пункте 4.

Листинг 5.10 Функция приема сообщения от клиента.

```
def recv_message(server):
    return server.recv(1024).decode
```

6 СТАТИСТИЧЕСКИЕ ТЕСТЫ ДЛЯ ПОТОЧНЫХ ШИФРОВ

6.1 Обоснование актуальности тестов

Поточные шифры построены на имитации концепции одноразового блокнота. В отличие от одноразового блокнота, эти шифры не являются достаточно надежными, однако, их намного удобнее использовать на практике в виду использования в их алгоритме ключа для генерации псевдослучайной последовательности. Такая последовательность должна быть неотличима от совершенно случайной последовательности. Данное утверждение означает, что последовательность должна иметь одинаковое количество нулей и единиц, равновероятность распределения символов, а также отсутствие в последовательности статистических взаимосвязей.

Для того, чтобы определить, насколько полученная последовательность близка к совершенно случайной последовательности, в криптоанализе используются такие методы исследования, как статистическое тестирование.

Основным принципом, на котором строятся статистические тесты является проверка нулевой гипотезы H_0. Нулевая гипотеза утверждает, что тестируемая последовательность является совершенно случайной. Альтернативной гипотезой H_1 является гипотеза о том, что последовательность случайной не является. По результатам статистического тестирования нулевая гипотеза либо подтверждается, либо отвергается.

Генератор, который подлежит тестированию, производит двоичные последовательности фиксированной длины. Выполняется набор статистических тестов, каждый из которых оценивает последовательность по определенным критериям и вырабатывает значение p-value. На основе полученных значений может быть сделано заключение о качестве тестируемой последовательности. [5]

В результате прохождения тестов может быть три варианта исходов:

1) Анализ не показал отклонение от случайности
2) Анализ ясно указал на отклонение от случайности
3) Анализ является недостаточным для выводов

Наличие хороших статистических свойств является одним из наиболее важных критериев надежности алгоритма поточного шифрования.

6.2 Исследование реализации

Для исследования реализации алгоритма шифрования были выбраны тексты содержащие русский язык. Сервер хранит 600 текстов, которые будут служить файлами для шифрования клиентов. Сервер выбирает 128 файлов и передает клиентам. Клиенты шифруют исходные сообщения и записывают их в файлы. Анализ содержания файлов проводится статистическим тестом «Стопка книг». Для того, чтобы дать точные значения тестов были проведены 100 сессий. Входные данные для тестирования: алфавит будет состоять из 256 символов, верхняя часть 128 элементов, нижняя 128.

На вход будут подаваться файлы форматов pdf, elf, docx, jpg, zip, txt. Каждый файл пройдет тестирование, состоящее из 100 сессий шифрования. На каждой сессии 6 файлов будут шифроваться одним набором из 128 файлов. На каждой сессии наборы будут обновляться.

Шифрование с помощью текстов содержащих только русский алфавит показал плохие результаты. Причина таких результатов в избыточности содержимого данных текстов.

Таблица 6.1 – Результаты тестирования файлов, содержащих тексты русского языка.

	Удачно	Неудачно
elf	21%	89%
docx	75%	25%
jpg	77%	23%
pdf	43%	57%
zip	57%	43%
txt	58%	42%

Из 256 символов алфавита в данном тесте встречаются 45 символов. Данный набор текстов не подходит для шифрования сообщений из-за большого процента проваленных тестов, что означает, что не достигнута вероятность следующего символа 1/2.

Во втором случае сервер хранит 600 файлов расширения djvu и jpg. Данный выбор позволит снизить избыточность файлов для шифрования,

из которых будет формироваться шифротекст. Условия теста такие же как и в первом случае, были проведены 100 сессий шифрования.

В данном случае, реализация шифра показала результаты:

Таблица 6.2 – Результаты тестирования файлов расширения djvu и pdf.

	Удачно	Неудачно
elf	82%	18%
docx	92%	8%
jpg	83%	17%
pdf	75%	25%
zip	78%	22%
txt	66%	34%

Из таблицы можно сделать вывод, что в данном случае распределение символов в тексте близкое к равномерному так как бинарные файлы улучшили показатели по сравнению с первым тестовым набором. Распределение символов в файлах относительно алфавита в этот раз охватывает 176 символов из 256, что является недостаточным для шифрования.

В третьем случае сервер хранит 600 файлов, заполненных псевдослучайными значениями. Каждый символ сгенерированных файлов будет иметь алфавит длинной 256 и включать в себя каждый символ алфавита. Данные символы распределены равномерно.

Таблица 6.3 – Результаты тестирования сгенерированных файлов.

	Удачно	Неудачно
elf	85%	15%
docx	84%	16%
jpg	88%	12%
pdf	76%	24%
zip	82%	18%
txt	83%	17%

Из таблицы видно, что теперь процент удачных тестов среди 6 расширений файлов близкое друг к другу. Но количество неудачных превышает 12%, что является для шифра плохим результатом. При увеличении количества тестов, процент неудачных тестов будет расти, соответственно вероятность, что злоумышленнику попадется файл с неравномерным распределением случайности битов файла.

Рассмотрим конфигурацию, когда на сервере находятся файлы расширения rar и zip в количестве 600 файлов.

Таблица 6.4 – Результаты тестирования файлов rar и zip.

	Удачно	Неудачно
elf	92%	8%
docx	91%	9%
jpg	91%	9%
pdf	98%	2%
zip	95%	5%
txt	31%	69%

После проведенных тестов бинарные файлы показали лучшие результаты среди прошлых тестов. При шифровании docx и jpf из 100 тестов всего по 9 текстов провалили тесты. Но плохие результаты показали файлы расширения txt. Данная ситуация произошла из-за того, что rar и zip файлы содержат служебные части форматов, что при операции xor дает исходное сообщение. Входное сообщение txt составляет 30 символов, что попадает в зону служебной части файлов для шифрования. На выходе получалась практически исходная последовательность.

Данная особенность всех бинарных файлов также позволяет утверждать, что при избавлении от служебной части файлов для шифрования можно добиться лучших результатов.

Для решения проблемы из прошлого теста очистим служебную часть в каждом файле, который будем передавать клиентам. В данном тесте будут использоваться файлы из прошлого теста без служебной части, что позволит улучшить результаты шифрования.

Таблица 6.5 – Результаты тестирования файлов rar и zip с удаленной служебной частью.

	Удачно	Неудачно
elf	96%	4%
docx	95%	5%
jpg	96%	4%
pdf	98%	2%
zip	96%	4%
txt	96%	4%

В данном случае процент удачных тестов достиг минимум 95%. Также txt файлы теперь находятся наравне с другими файлами. Вероятность распределения в файлах для шифрования равномерны.

ВЫВОД

Использование списком файлов с русским языком, привел к плохой защищенности шифротекста так как вероятность следующего бита шифротекста не была равна 1 / 2. Для elf файлов данный тест показал самые плохие результаты выдав 89% не пройденных тестирований из 100%.

При рассмотрении шифрования используя djvu и jpg файлы, показатели тестов были улучшены, но тестов, проваливших тестирование всё также много.

Из приведенных результатов можно сделать вывод, что реализация шифра максимально устойчива в случае, если использовать тексты, состоящие из rar и zip архивов без служебной части. В данном тесте получился самый малый процент не пройденных тестирований шифрованных последовательностей.

ЗАКЛЮЧЕНИЕ

Было проведена разработка и исследование характеристик реализации шифра Шеннона. Благодаря успешному подбору файлов шифрования реализация обеспечивает защищенный канал связи между клиентами и гарантию того, что перехваченное сообщение не будет расшифровано за реальное количество времени. Данная реализация позволяет общение клиентов в режиме передачи сообщений либо файлов. Разработанная архитектура клиент-серверного приложения может использоваться в разработке чатов, файловых хранилищ.

Обязательными условиями реализации были выявлены:

1) Закрытый ключ должен быть длины 128 бит.
2) Файлы ключи должны состоять из файлов расширения rar и zip.
3) Рекомендуемый размер файлов для шифрования 128Кб, максимальный размер комплекта файлов будет составлять 16Мб.

В дальнейшем планируется исследование возможностей использования шифра для трансляции аудио/видео потока от одного клиента второму клиенту либо множеству клиентов. Что позволит проводить защищенные конференции, звонки, осуществлять удаленное управление.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Безопасность сетей / Э. Мэйволд – М.: Национальный Открытый Университет «ИНТУИТ» — 2006. — 528 с
2. Клод Шеннон. Теория связи в секретных системах / пер. с англ. В. Ф. Писаренко // Работы по теории информации и кибернетике / Под редакцией Р. Л. Добрушина и О. Б. Лупанова. — М. : Издательство иностранной литературы, 1963. — 829 с.
3. Защита программ и данных. / Проскурин В.Г. Издательство: М.: Академия, второе издание, 2012 г. 208 с.
4. Бабаш А. В., Гольев, Ю. И., Ларин Д. А. и др. Криптографические идеи XIX века // Защита информации. Конфидент — СПб.: 2004. — вып. 3.
5. Е.В. Игоничкина Статистический анализ поточных шифров [Электронный ресурс] // URL: https://cyberleninka.ru/article/v/statisticheskiy-analiz-potochnyh-shifrov (дата обращения 10.05.2017)
6. Криптографическая защита информации: учебное пособие/ А.В.Яковлев,А.А.Безбогов,В.В.Родин,В.Н.Шамкин.–Тамбов: Изд-во Тамб.гос.техн.ун-та, 2006.– 140с. – 100экз. – ISBN 5-8265-0503-6.
7. Рябко Б.Я, Ракитский А.А Теоретико-информационный анализ шифров Шеннона. [Электронный ресурс] / Рябко Б.Я, Ракитский А.А URL: https://eprint.iacr.org/2016/429.pdf
8. Лутц М. Изучаем Python, 4-е издание. – Пер. с англ. – СПб.: Символ-Плюс, 2011. – 1280 с.
9. А.В.Яковлев Криптографическая защита информации : учебное пособие / А.В.Яковлев, А.А. Безбогов, В.В. Родин, В.Н. Шамкин. – Тамбов : Изд-во Тамб. гос. техн. ун-та, 2006. – 140 с.
10. Стасев Ю.В. Исследование методов криптоанализа поточных шифров [Электронный ресурс] / Ю.В. Стасев, А.В. Потий, Ю.А. Избенко. – Режим доступа к ресурсу: http://www.nrjetix.com/fileadmin/doc/publications/articles/stasev_potiy_izbenko_ru.pdf
11. Алферов А.П Основы криптографии / А.П. Алферов, ., Зубов А.Ю., Кузьмин А.С., Черемушкин А.В. – М.: Гелиос АРВ, 2005. – 480 с.

Приложение А.

Листинги

Листинг А.1 - Сервер

```
import socketserver
import random
import time
import os


class Shannon(socketserver.BaseRequestHandler):
    def handle(self):
        self.P = 0
            self.G = 0
            self.Zb = 0
        self.Q = 0
        self.DiffieHellman()

        self.downloadFiles()
        self.sendFiles()

    def sendFiles(self):
        list = self.selection()
        for text in list:
            with open("textCode/" + text, "rb")as f:
             line = f.read(262144)
             lenText = os.path.getsize("textCode/"+text)
             self.request.send(line)
             self.request.send(b"next")

    def DiffieHellman(self):
        self.Q = random.randint(0, (2 ** 128) - 1)
        while self.miller_rabin(self.Q) == False or self.miller_rabin(2 * self.Q + 1)
== False:
            self.Q = random.randint(0, (2**128)-1)
        self.P = 2 * self.Q + 1
        self.G = random.randint(0, (2**128)-1)
        while (self.powmod(self.G, self.Q, self.P) == 1):
            self.G = random.randint(0, (2 ** 128) - 1)
        self.Xa = random.randint(0, (2 ** 128) - 1)
        self.Ya = self.powmod(self.G, self.Xa, self.P)
        self.request.send(str(self.P).encode())
        time.sleep(0.2)
```

```
    self.request.send(str(self.G).encode())
    time.sleep(0.2)
    self.request.send(str(self.Ya).encode())
    time.sleep(0.2)
    self.Yb = int(self.request.recv(1024).decode())
    self.Zb = self.powmod(self.Yb, self.Xa, self.P)

def powmod(self, a, step, mod):
    b = 1
    while step:
        if (step&1) == 1:
            step -= 1
            b = (b * a) % mod
        step >>= 1
        a = (a * a) % mod
    return b % mod

def miller_rabin(self, n, s=50):
    for j in range(1, s + 1):
        a = random.randint(1, n - 1)
        b = self.Tos(n - 1)
        d = 1
        for i in range(len(b) - 1, -1, -1):
            x = d
            d = (d * d) % n
            if d == 1 and x != 1 and x != n - 1:
                return True
            if b[i] == 1:
                d = (d * a) % n
                if d != 1:
                    return True
                return False

def Tos(self, n):
    r = []
    while (n > 0):
        r.append(n % 2)
        n = n / 2
        return r

def selection(self):
    list = os.listdir("textCode/")
    lenList = len(list)
```

```
        ans = []
        while 1:
            select = random.randint(0,lenList-1)
            if (list[select] in ans) == False:
                ans.append(list[select])
            if len(ans) == 128:
                return ans

if __name__ == '__main__':
    HOST, PORT = "0.0.0.0", 1337
    server = socketserver.TCPServer((HOST, PORT), Shannon)
    server.serve_forever()
```

Листинг А.2 - Клиент

```
import socket
import random
import os
import time
import base64


def generationKey(server):
    P = int(server.recv(1024).decode())
    G = int(server.recv(1024).decode())
    Ya = int(server.recv(1024).decode())
    Xb = random.randint(0, (2 ** 128) - 1)
    Yb = powmod(G, Xb, P)
    server.send(str(Yb).encode())
    return powmod(Ya, Xb, P)

def powmod(a, step, mod):
    b = 1
    while step:
        if (step & 1) == 1:
            step -= 1
            b = (b * a) % mod
        step >>= 1
        a = (a * a) % mod
    return b % mod
```

```
def downloadFiles(server):
    data = ""
    for i in range(0, 128):
        with open("text/"+str(i)+".txt", "wb”) as f:
            while True:
            data = server.recv(262144)
            if data == b"next":
              break
            else:
              file.write(data)
            if not data:
              break

def bits(n):
    ans = []
    sum1 = 0
    for i in range(0, 128):
        a = bit(n, i)
        if a == 1:
            sum1 += 1
        ans.append(a)
    return ans

def bit(num, pos):
    return (num & (1 << pos)) >> pos


def xor_message(text1, list):
    for i in range(0, 128):
        if list[i] == 1:
            with open("text/" + str(i) + ".txt", "rb") as f:
                        text1 = xor(text1, bytearray(f.read()))
    return text1

def xor(text1, text2):
    lenText2 = len(text2)
    j = 0
    for i in range(len(text1)):
        text1[i] = text1[i] ^ text2[j]
        j += 1
        if j == lenText2-1:
            j = 0
    return text1
```

```
if __name__ == '__main__':
    for i in range(1, 100):
        HOST, PORT = "127.0.0.1", 1337
        server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        server.connect((HOST, PORT))
        Z = generationKey(server)
        ans = bits(Z)
        with open('test/statistic.txt', 'a') as f:
            f.write('{}\n'.format(sum(ans)))
        downloadFiles(server)
        server.close()
        for name in os.listdir("files/"):
            with open("files/{0}".format(name), "rb") as f:
                text_message = bytearray(f.read())
            # Шифрование исходного
            text1 = xorTest(text_message, ans)
            with open('test/{0}_{1}.txt'.format(i, name), 'wb') as f:
                f.write(text1)
                print("{}_{} encode success".format(i, name))
            with open('test/{0}_{1}.txt'.format(i, name), 'rb') as f:
                text1 = bytearray(f.read())
            # Шифрование шифртекста
            text1 = xorMessage(ans, text1)
            with open('success/{0}_{1}'.format(i, name), 'wb') as f:
                f.write(text1)
                print("{}_{} decode success".format(i, name))
```

Листинг А.3 - Парсер

```
import re
import requests
import wget
import os
from pyunpack import Archive


def req():
    site = "http://www.knigitxt.com/"
    for i in range(1, 31):
```

```
            data = requests.get("http://www.knigitxt.com/authors/all/{}".format(i))
            authors = re.findall("/author/\d{1,5}", data.text)
            for author in authors:
                data = requests.get("http://www.knigitxt.com{}".format(author))
                books = re.findall("/preview/[A-Za-z0-9_()-]{1,350}.html", data.text)
                for book in books:
                    data = requests.get("http://www.knigitxt.com{}".fortmat(book))
                    downloads = re.findall("/download/\d{1,10}.html", data.text)
                    for download in downloads:
                        data =
requests.get("http://www.knigitxt.com{}".format(download))
                        rars = re.findall("/media/\w{1,100}/[A-Za-z0-9_%()-
]{1,100}/[A-Za-z0-9_%()-]{1,100}.rar", data.text)
                        for rar in rars:
                            file = wget.download("http://www.knigitxt.com"+rar)
                            Archive(file).extractall('text/')
                            os.remove(file)


if __name__ == '__main__':
    req()
```

Листинг А.4 – Вспомогательные функции

```
import os
import hashlib
import random

def rename():
    for name in os.listdir("text/"):
        newName = hashlib.md5()
        newName.update(name.encode())
        os.rename("text/"+name, "textCode/"+newName.hexdigest()+".txt")


def delete_space():
    for name in os.listdir("textCode/"):
        with open("textCode/{}".format(name), "rb") as f:
            text = bytearray(f.read())
        with open("textCode/{}".format(name), "wb") as f:
            f.write(text[300::])
```

```
def generation():
    for i in range(1, 301):
        s = bytearray(random.randint(0, 255) for j in range(0, 200000))
        with open('backups/generation/{}'.format(i), 'wb') as f:
            f.write(s)
            print('write {}'.format(i))

def del_size():
    for name in os.listdir("textCode/"):
       if os.path.getsize('textCode/{}'.format(name)) < 256000:
           os.remove('textCode/{}'.format(name))


if __name__ == '__main__':
    a = input('Input')
    if a == '1':
        rename()
        delete_space()
        del_size()
    elif a == '2':
        delete_space()
    elif a == '3':
        generation()
    elif a == '4':
        del_size()
```