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

Кафедра ПМ и К

Допустить к защите

Зав.каф. _____ *Фионов А.Н*

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

Реализация и исследование поточного шифра на основе алгоритма Шеннона.

Магистерская диссертация

по направлению 09.04.01 «Информатика и вычислительная техника»

Студент Чусовитин А.Р /……/

Руководитель Ракитский А.А / ……..……/

Новосибирск 2019 г.

Содержание

Введение

В настоящее время шифры, используемые для осуществления безопасного соединения по сети и передачи данных, обладают рядом недостатков. Некоторые реализации шифра RC4, используемого, например, в протоколе TLS, были скомпрометированы и не могут считаться достаточно безопасными. Кроме того, актуальным является вопрос снижения трудоёмкости вычисления ключей и шифрования данных без потери защищённости. В работе [1] Рябко Б.Я. были рассмотрены два шифра Шеннона [2] и была строго доказана их теоретическая надёжность, а также указаны минимальные значения параметров, необходимых для достижения такой надёжности. Эти шифры отличаются простотой реализации и, соответственно, эффективностью работы, поэтому их можно рассматривать в качестве альтернативы существующим методам шифрования передаваемых данных. В данной работе рассматривается один из шифров Шеннона, авторами предлагается метод использования данного шифра для защиты данных, передаваемых по сети.

Кроме того, на основе разработанного прототипа предлагаемого метода, проведено экспериментальное исследование защищённости рассматриваемого шифра при помощи общепризнанных статистических тестов. Полученные результаты позволяют говорить о возможности практического использования шифра Шеннона для защиты передаваемых по сети данных и показывают высокую эффективность такой защиты

1. Криптография

1.1. Криптографические алгоритмы

Криптография — наука, занимающаяся методами преобразования информации, не позволяющим противнику извлечь данные из перехватываемых им сообщений. Криптографический алгоритм — система преобразований входной информации в шифртекст 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:

- Симметричный метод, использующий один и тот же ключ, как для шифрования, так и для дешифрования
- Ассиметричный метод, использующий для тех же целей два различных ключа.

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

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

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:

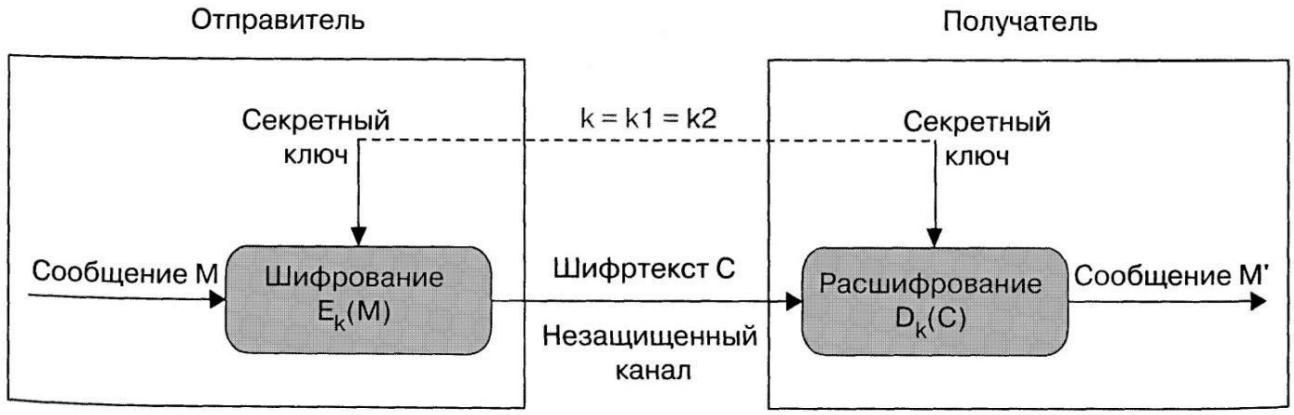

Рисунок 1.1 – Система симметричной криптосистемы шифрования

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 шифртекста. 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 шифртекст C.
4. Расшифровка происходит операцией XOR между гамма-последовательностью и шифртекстом 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. Синхронный поточный шифр

Синхронные поточные шифры – такие шифры в которых поток ключей генерируется независимо от исходного текста и шифртекста.[3]

Во время шифрования гамма-последовательность выдает биты для шифрования исходного сообщения, которые одинаковы с битами для расшифровывания. Потеря знака шифртекста 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 шифртекст специальных маркировок. Благодаря этому не правильное расшифровывание передаваемого шифртекста 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.

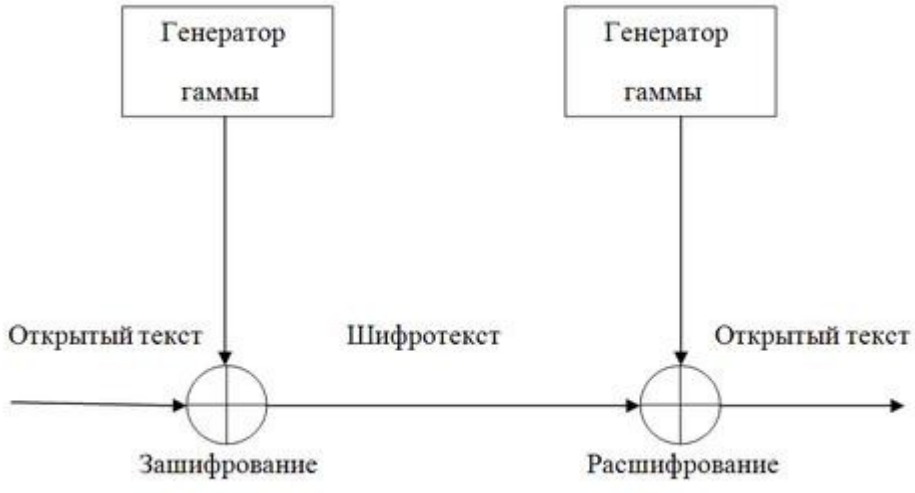

Рисунок 1.2 – Схема синхронного поточного шифрования

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 шифртекста 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 шифртекста.

Однако, синхронные потоковые шифры имею уязвимость к атакам на основе изменения отдельных бит данных шифротекста. Злоумышленник может изменить эти отдельные биты так, что шифротекст 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.

Соответственно этим подходам также были указаны и теоретические критерии Райнера Рюппеля для проектирования поточных систем:

1. Длинные периоды выходных последовательностей
2. Большая линейная сложность
3. Диффузия – рассеивание избыточности в подструктурах
4. Каждый бит потока ключей должен быть сложным преобразованием большинства битов ключа
5. Критерий нелинейности для логических функций.

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.

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.

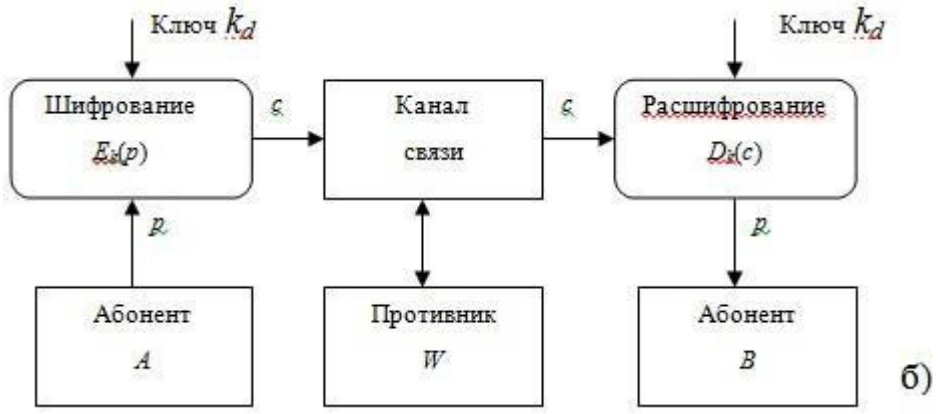

Рисунок 2.1 – Схема абсолютно стойкой системы

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 шифртекста.

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.

2.4. Статистические тесты для поточных шифров

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:

- Анализ не показал отклонение от случайности
- Анализ ясно указал на отклонение от случайности
- Анализ является недостаточным для выводов

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

3. Шифр Вернама

3.1. Описание шифра

Шифр Вернама, изобретённый 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 шифртекст (110100). Для того, чтобы получить исходное сообщения (100110) из полученного шифртекста достаточно применить обратную операцию исключающее ИЛИ к принятому шифртексту (110100) и закрытому ключу (010010).

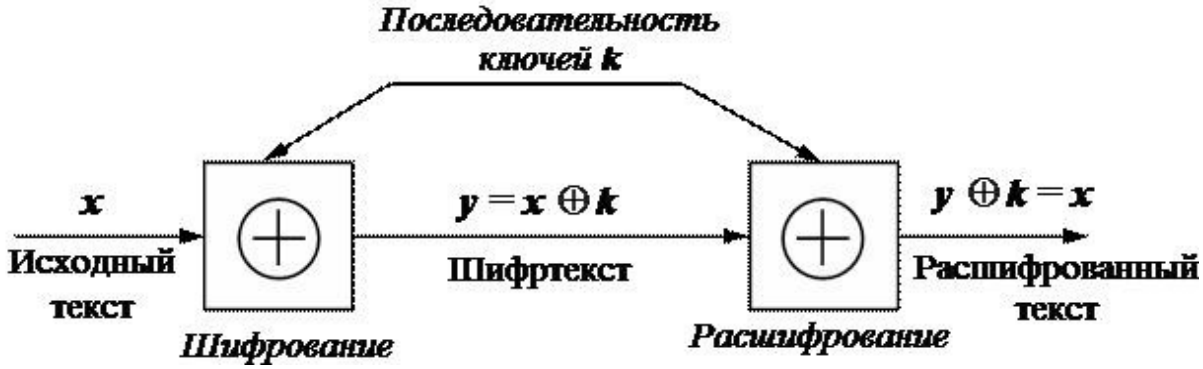

Рисунок 3.1 – Схема шифрования алгоритмом Вернама

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:

   $m'_1 = m_1 \oplus c$

   $m'_2 = m_2 \oplus c$

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

2) 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.

3) 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.
4) Длина гамма-последовательности не должна быть короче длины исходного сообщения, так как при выборе гамма-последовательности меньшей длины можно будет подобрать размер, проанализировать блоки шифртекста и подобрать биты гамма-последовательности.

3.3. Недостатки

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 шифртекста 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.

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.

Применение повторного ключа влечет за собой подбор закрытого ключа и последующее вскрытие шифра.

4. Шифр RC-4

4.1. Описание шифра

Алгоритм поточного шифрования RC-4 строится на генераторе гамма последовательности. Алгоритм состоит из функции генератора гаммы, который выдает последовательность битов ключа.

Алгоритм шифрования:

- Функция генерирует поток битов ключа.
- Каждый бит генерируемой последовательности складывается по модулю два с открытым текстов. В результате получается шифртекст.

Алгоритм расшифровки:

- Повторно функция генерирует поток битов ключа.
- Шифртекст побитово складывается с результатом функции генератора. На выходе получается исходный текст.

RC-4 – относится к классу алгоритмов, определяемые размером блока (S-блок). Параметр n – это размер слова, он определяет длину S-блока. Зачастую n=8, но для повышения криптостойкости необходимо увеличивать параметр n. При увеличении n время начальной итерации увеличится, но скорость шифрования возрастет.

Внутри состояние алгоритма представляется в виде массива длиной 2^n и пары счетчиков. Массив – это S-блок и он всегда содержит перестановку 2^n всех возможных значений слова. Инициализация алгоритма состоит из:

1) Инициализации S-блока.
2) Генерации псевдослучайного слова K.

Инициализация S-блока также известна как «key-scheduling algorithm». В алгоритме используется ключ, который знает пользователь. Ключ имеет длину L байт. Инициализация начинается с заполнения S-блока, далее массив перемешивается с помощью перестановок. В S-блоке хранится всегда исходный набор данных. Формально инициализацию S-блока можно представить:

Листинг 4.1 Формирование S-блока

```
for i in range(255):
    s[i] = i
for i in range(255):
    j = (j + s[i] + key[i % L]) % 256 # n = 8. 2**8 = 256 s[i], s[j] = s[j],
    s[i]
```

Генерация псевдослучайного слова K также называется «pseudo-random generation algorithm». Генератор выбирает значения из S-блока. В одном цикле определяется одно n-битное слово K из ключевой последовательности. Затем ключевое слово будет сложено по модулю 2 с исходным текстом.

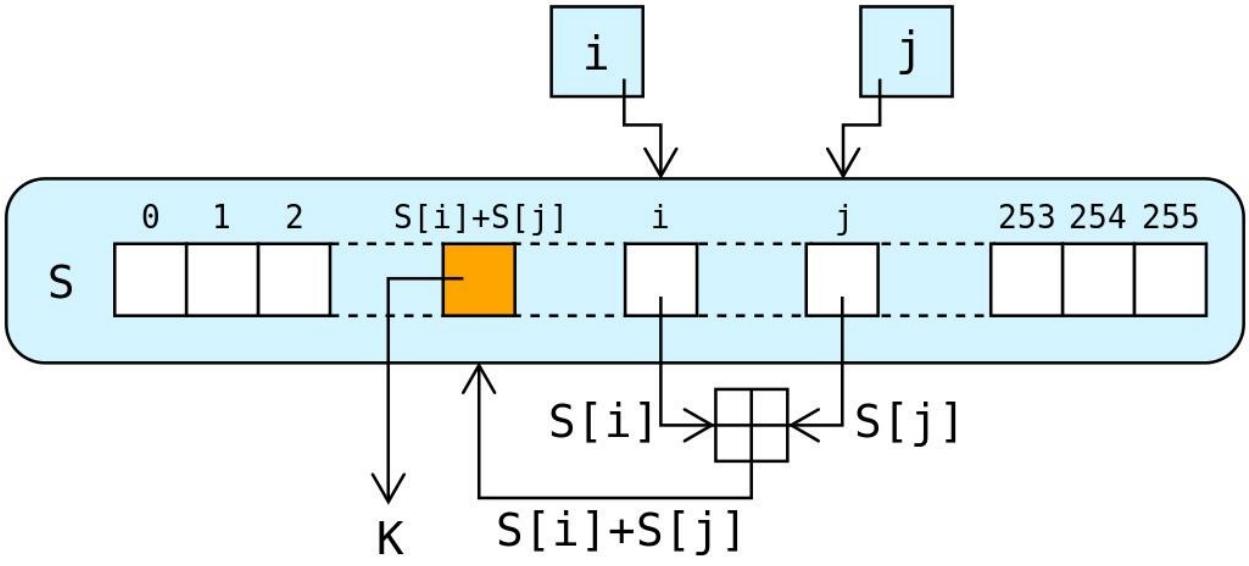

Рисунок 4.1 – Генератор ключевого потока RC4

4.2. Криптографическая стойкость

В отличие от современных шифров, RC-4 не использует подход «число, которое может быть использовано один раз» (nonce) вместе с ключом. Отсюда следует, что, если один ключ должен использоваться долгое время, тогда сама криптосистема, использующая алгоритм RC-4, должна комбинировать оказию и ключ для получения потокового ключа. Один из выходов – генерировать новый ключ с помощью хеш-функции от ключа и nonce. Из-за этого алгоритм может стать уязвимым.

В 1995 году Андрю Руз экспериментально обнаружил, что первый байт ключевого потока коррелирован с первыми тремя байтами ключа, а первые несколько байт перестановки после алгоритма расписания ключей коррелированы с некоторой линейной комбинацией байт ключа. В 2007 году Пол, Рафи и Майтрэ доказали коррелированность ключа и ключевого потока.

Летом 2015 года Мэти Ванхоф (Mathy Vanhoef) и Франк Писсенс (Frank Piessens) из университета Левена в Бельгии продемонстрировали реальную атаку на протокол TLS, использующий RC4 для шифрования передаваемых данных.
Идея взлома базируется на принципе MITM

4.3. Недостатки

Алгоритм RC-4 довольно уязвим, если:

- Используется не случайные или связанные ключи
- Одна ключевая последовательность используется дважды.

Из-за этих недостатков, алгоритм может сделать криптосистему небезопасной.

5. Шифр Шеннона

5.1. Описание шифра

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

В криптосистему с бегущим ключом входит система Шеннона. Шеннон предложил два шифра с бегущим ключом, которые простые в реализации и надежные, но не исследовал их свойства. В первом шифре Шеннон предложил, что вместо одного текста на английском языке использовать D различных, прибавляя их к исходному сообщению. При достаточно большом количестве ключей повысится надежность шифра. Но Шеннон не привел никаких доказательств данного шифра. Во втором шифре Шеннон предложил брать каждую N-ую букву текста пропуская остальные буквы, тем самым выбор каждого следующего символа будет практически независим. Этот метод также должен был дать надежный шифр.

В работе рассмотрен первый шифр Шеннона. Исследованы его свойства и реализация криптосистемы на основе шифра.

5.2. Теоретический материал

Определим шифры с бегущим ключом следующим образом: пусть исходное сообщение $M_1 \ldots M_t$, ключ $K_1 \ldots K_t$ и шифротекст $E_1 \ldots E_t$ – это последовательности символов из алфавита A={0,1,…,n-1}, где n≥2. Будем считать, что шифрование и дешифрование определено правилами

$$E_i = c(M_i,K_i),\ \ i = 1\ldots t \qquad (5.1)$$

$M_i = d(E_i,K_i)$, $i = 1\ldots t$, таким образом $d(e(M_i,K_i),K_i) = M_i$. (5.2)

В данной работе рассматривается первый шифр Шеннона, поэтому необходимо представить его формальное описание. Пусть есть s источников

$M,K_1,K_2,\ldots,K_{s-1},s \geq 2$ и каждый из источников или генерирует символы из алфавита = {0,1,…, − 1}. Будем считать, что - это сообщение, а $_1,\ldots,_{-1}$ последовательности ключей. Тогда шифротекст задаётся следующим образом:

$$E_i = C(M_i, K_i) = M_i + K_i^1 + K_i^2 + \ldots + K_i^{s-1} \pmod n \qquad (5.3)$$

где i – номер символа. Дешифрование в таком случае очевидно и будет совпадать с шифрованием.

Пусть есть два абонента Алиса (А) и Боб (Б). Абонент А хочет передать сообщение М абоненту Б таким образом, чтобы никто посторонний его не смог прочесть. Пусть у обоих абонентов есть одинаковый набор из 1024 файлов, каждый размером от 1 до 8 Кбайт. Эти файлы будут использоваться в качестве ключей шифрования, именно с ними передаваемое сообщение необходимо складывать по модулю 2 (в нашем случае, считаем, что все источники генерируют последовательности из алфавита = {0,1}). Очевидно, что любое сообщение можно представить в двоичном виде, поэтому указанное допущение является верным.

В случае, когда длина передаваемого сообщения больше длины ключевых файлов, необходимо эти файлы расширить, в таком случае мы будем дублировать файл. Т.е. $E' = E|E$, где Е – ключевой файл, а операция | - это конкатенация. В таком случае, необходимо выбрать файлы таким образом, чтобы их размеры были попарно взаимнопростыми числами. Необходимо объяснить это требование: если складывать по модулю 2 файлы, которые расширяются за счёт дублирования, то длина цикла в полученной сумме будет равна НОК($E_1,\ldots E_s$), и в случае, когда длины файлов являются взаимнопростыми, это значение будет равно:

$$НОК(E_1,\ldots E_s) = |E_1| \times \ldots \times |E_s| \qquad (5.4)$$

Таким образом, даже при 4-х ключевых файлах, мы получим последовательность с длиной цикла не менее 1012 байт, что вполне достаточно для передачи любого реального сообщения или потока данных. Это требование

необходимо, т.к. наличие цикла в ключе может быть выявлено при помощи статистических тестов и будет являться серьезной слабостью для любого шифра.

Отдельно стоит отметить, как именно формируются ключевые файлы. Эти файлы являются архивами, полученными, например, при помощи архиватора rar, без служебной части. Такие форматы ключевых файлов были сформированы в результате проведённого исследования, которое показало, что статистические тесты не могут выявить в них какой-либо закономерности, и считают подобные файлы случайными последовательностями.

Алиса и Боб формируют общий ключ, например, при помощи системы ДиффиХеллмана[3]. Сформированный закрытый ключ является стартовым значением для генератора псевдослучайных чисел. Установив стартовое значение генератора, Алиса и Боб получают число N – количество файлов, которые будут участвовать в формировании ключа. Число N принимает значения от 4 до . После чего генерируются чисел – номера используемых ключевых файлов.

Необходимо пояснить, как влияет число S на трудоёмкость взлома шифра. В данном случае мы считаем, что и абоненты, и злоумышленник имеют изначально одинаковый набор файлов и у злоумышленника есть полностью перехваченное зашифрованное сообщение E. В таком случае, чтобы узнать значение исходного передаваемого значения M злоумышленнику необходимо узнать какие именно файлы использовались в качестве ключевых. С учётом «случайности» зашифрованной последовательности, злоумышленнику необходимо будет перебрать все возможные наборы файлов, использованных в качестве ключевых. Трудоёмкость такого перебора определяется следующей формулой:

$$T = C_{1024}^{N} = \frac{1024!}{N!(1024-N)!}, \qquad (5.5)$$

При N=20, $T \sim 2^{139}$, при N = 40, $T \sim 2^{240}$.

Таким образом, при относительно небольшом значении s мы можем добиться достаточно высокой трудоёмкости перебора всех возможных ключей. В нашем случае это позволяет довольно легко найти компромисс между скоростью формирования ключа и защищённостью системы. Очевидно, что при 20 ключевых

файлах задача подбора ключа является не решаемой. Однако, чтобы соответствовать современным требованиям безопасности, рекомендуется использовать N ≥40 ключевых файлов.

Отдельно необходимо отметить, что операция побитового исключающего или (XOR, или сложение по модулю 2) является легко векторизируемой на современных процессорах, а, значит, позволяет складывать за 1 такт вычислительного ядра процессора блоки по 512 бит. Это позволяет существенно ускорить процесс формирования ключа и шифротекста, что положительно сказывается на производительности предлагаемого метода.

5.3. Инициализация

Чтобы создать защищенный канал общения между двумя пользователями необходимо снабдить клиентов файлами ключей. Для соединения двух клиентов у обоих клиентов должны быть 1024 файла, которые в дальнейшем будут образовывать шифрключ. 1024 файла передаются абонентам по открытым каналам связи. Важно отметить, что длины файлов варьируются от 1024 до 4096 байт. При этом, длины файлов должны удовлетворять правилу:

$$НОК(F_1,F_2,\ldots,F_n) = |F_1| * \ldots * F_n \qquad (5.6)$$

Пользователи должны знать закрытый ключ, который к примеру, должен быть рассчитан алгоритмом Диффи-Хеллмана. Ключ должен быть достаточно большим числом, чтобы третье лицо не могло подобрать его. К примеру 2^{1024}. Это закрытое число, пользователи используют в качестве ключа для конфигурации генератора псевдослучайных чисел, обеспечивая одинаковую последовательность генерируемых чисел. Эти числа, являются номерами файлов, используемых для шифрования.

Листинг 5.1 – Выбор файлов для шифрования

```
def _selection(self, private_key):
    """
    Метод для выбора файлов сессии и файлов для шифрования
    :param private_key:
```

```
        :return:
        """
        seed(private_key)
        self._selection_files()
        self._selection_files_code()

    def _selection_files(self):
        """
        Метод для формирования множества файлов для сессии
        :return:
        """
        file_list = listdir("%s\\text" % self.helper_dir)
        if len(file_list) < self.count_files:
            raise ('Ошибка', 'Нужно больше файлов для шифрования')
        self.files = set()
            while len(self.files) < self.count_files:
            self.files.add(choice(file_list))
        self.files = list(self.files)

    def _selection_files_code(self):
        """
        Метод выбирает файлы, которыми будет использовать для
        шифрования :param private_key:
        :return:
        """
        count_files = self.count_files_key
        self.files_code = set()
        while len(self.files_code) < count_files:
            self.files_code.add("%s\\text\\%s" % (self.helper_dir,
choice(self.files)))
```

На основе выбранных файлов происходит формирование ключа для шифрования. Ключ представляет из себя гамму, которая имеет достаточно большую длину, для шифрования больших файлов. Тут возможно 2 режима работы шифра.

- Ключ формируется максимальной длинны.
- Ключ формируется во время шифрования.

В первом случае, ключ находится в памяти, но обеспечивает высокую скорость шифрования. Во втором случае, ключ рассчитывается во время шифрования исходной последовательности, что приводит к снижению скорости шифрования. В рамках работы, рассматривается 1 подход, когда храним ключ в памяти.

Листинг 5.2 – Формирование ключа

```
def _make_code_key(self):
    """
    Метод формирует ключ шифрования
    :return:
    """
    for file in self.files_code:
        with open(file, 'rb') as f:
            code = bytearray(f.read())*50
            if self.key:
                self.key, code = (code, self.key) if len(code) > len(self.key)
else (self.key, code)
                self.key = self._xor(self.key, code)
            else:
                self.key = code

@staticmethod
def _xor(text, code):
    """
    Метод выполняет операцию XOR к 2 последовательностям
    bytearray :param text:
    :param code:
    :return:
    """
    for index in range(len(text)):
        text[index] = text[index] ^ code[index % len(code)]
    return text
```

5.4. Алгоритм шифрования

Когда пользователи провели инициализацию, первый пользователь приступает к этапу шифрования сообщения, используя рассчитанный при инициализации ключа шифрования. Процесс шифрования построен таким образом, что основной этап работы сводится выполнению «сложения по модулю 2» между исходным сообщением и ключом шифрования. После шифрования, пользователь отправляет зашифрованное сообщение второму пользователю. [7]

Второй пользователь к полученному зашифрованному сообщению применяем ключ шифрования. Ключ шифрования получился по тому же алгоритму, что и у первого пользователя.

6. Сравнение алгоритмов

6.1. Статистические тесты

Поточные шифры построены на имитации концепции одноразового блокнота. Суть заключается в том, чтобы сформированный ключ для шифрования был неотличим от случайной последовательности. Это означает, что зашифрованная таким ключом последовательность должна иметь равную вероятность распределения 0 и 1, а также должны отсутствовать какиелибо статистические взаимосвязи. Теоретически было доказано, что рассматриваемый в данной работе шифр Шеннона будет обладать подобным свойством для текстов на английском языке, если в качестве ключевой последовательности используется не менее 4-х англоязычных текстов. Для экспериментального подтверждения надёжности шифра был выбран статистический тест «стопка книг» [4]. Для анализа была разработана программа для шифрования данных по описанному методу. При помощи данной программы шифруются файлы различных типов: exe, jpg и zip. Размеры файлов приведены в таблице 6.1.

Таблица 6.1 - Размеры исследуемых файлов

Формат файла	Размер
*.txt	0.2 Мб
*.exe	543 Мб
*.rar	16 Мб

В рамках данного исследования рассматривается по 10.000 сессий шифрования, каждый файл шифруется при помощи разработанного метода, а полученный шифротекст проверяется на случайность при помощи теста «Стопка книг». Результаты анализа приведены в таблице 6.2.

Таблица 6.2 - Результаты тестирования файлов

	Успешно, %	Неуспешно, %
*.txt	98	2
*.exe	96	4
*.rar	97	3

В данной таблице приводятся процентные данные по успешности/неуспешности прохождения теста при конфигурациях с 4 ключевыми файлами. Допустимым считается, если неуспех составил не более 5% проведённых тестов. Для проверки оптимального количества использования открытых текстов, которые используются для формирования ключа шифрования, проведем исследование изменяя количество этих файлов. Рассмотрим количество исследуемых файлов от 4 до 20 для шифрования exe файлов. Результаты анализа приведены в таблице 3.

Таблица 6.3 - Результаты тестирования при разных количествах ключевых файлов при шифровании exe файлов.

	Успешно, %	Неуспешно, %
4	95.190035	4.809965
5	94.808147	5.191853
6	95.144572	4.855428
7	94.989998	5.010002
8	95.099109	4.900891
9	95.269517	4.730483
10	95.125414	4.874586
11	95.151457	4.848543
12	95.080035	4.919965
13	95.196299	4.803701
14	95.118899	4.881101
15	95.431790	4.568210
16	95.369212	4.630788
17	95.190035	4.505632
18	94.808147	5.029764
19	95.144572	5.193992
20	94.989998	4.070131

Из результатов в данной таблице, можно определить, что минимальное количество ключевых файлов для шифрования можно использовать от 7 до 17 файлов. Однако для применения в современных криптосистемах необходимо провести тестирование для N≥40 ключевых файлов. В таблице 6.4 показаны результаты исследования статистических свойств при использование N≥40 ключевых файлов.

Таблица 6.4 - Результаты тестирования при разных количествах ключевых файлов.

	Успешно, %	Неуспешно, %
40	95.991984	4.008016
42	95.388778	4.611222
44	95.889780	4.110220
46	96.805897	3.194103
48	95.967742	4.032258
50	96.774194	3.225806
52	95.564516	4.435484
54	95.758065	4.241935
56	95.161290	4.838710
58	97.580645	2.419355
60	94.548387	5.451612
62	97.177419	2.822581
64	96.370968	3.629032
66	95.564516	4.435484
68	95.967742	4.032258
70	97.177419	2.822581
72	94.75	5.25
74	93.338710	6.661290
76	95.564516	4.435484
78	94.548387	5.451612
80	95.480549	4.519451

По результатам таблицы 6.4 можно сделать вывод, что статистические свойства зашифрованной последовательности сохраняются при увеличении количества ключевых файлов. Также при продолжении увеличения количества используемых ключевых файлов, статистические свойства будут сохраняться. Что позволяет использовать в рассматриваемой реализации применение любого количества файлов.

6.2 Сравнение скорости

Для сравнения скорости шифрования использовали файлы exe и rar расширений. Соответственно размерами из таблицы 1. Были проведены 10.000 сессий шифрования и усреднены результаты. Результаты приведены в таблице 4.

Таблица 6.5. Скорость шифрования

	Шеннон	RC4
*.exe	85 сек	652 сек
*.rar	188 сек	1537 сек

Из таблицы 2 можно сделать выводы, что реализация алгоритма Шеннона работает со скоростью 5,5 Мб/сек, а RC4 со скоростью 0.7 Мб/сек. Что показывает, реализация алгоритма Шеннона работает примерно в 8 раз быстрее алгоритма RC4.

7. Вывод

Метод базирующийся на одном из шифров Шеннона с доказанной теоретической криптостойкостью, может быть использован для создания защищенной криптосистемы. Это подтверждается теоретически [2], так и экспериментально полученными результатами статистического тестирования. В сравнение с текущим стандартом поточного шифра RC-4 скорость шифрования текущей реализации алгоритма Шеннона быстрее в 8 раз.

Криптосистема построенная на основе реализации шифра Шеннона может использоваться в реальных системах и являться альтернативой существующим поточным шифрам. Шифр не обладает большинством недостатков текущих шифров.

8. Заключение

Было проведена разработка и исследование характеристик реализации шифра Шеннона в сравнение с существующим стандартом поточного шифра RC-4. 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:

- Рекомендуемый размер файлов для шифрования 1 – 4 Кб, максимальный размер комплекта файлов будет составлять 4096 Кб.

9. Список использованных источников

1) Э. Мэйволд. Безопасность сетей. — 2006. — 528 с
2) Ryabko, B. (2018). Properties of two Shannon's ciphers. Designs, Codes and Cryptography, 86(5), 989-995
3) 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.
4) Проскурин В.Г. Защита программ и данных. Издательство: М.: Академия, второе издание, 2012 г. 208 с.
5) Бабаш А. В., Гольев, Ю. И., Ларин Д. А. и др. Криптографические идеи XIX века // Защита информации. Конфидент — СПб.: 2004. — вып. 3.
6) B. Ya. Ryabko, A. I. Pestunov, ""Book Stack" as a New Statistical Test for Random Numbers", Probl. Peredachi Inf., 40:1 (2004), 73–78; Problems Inform. Transmission, 40:1 (2004), 66–71
7) Ракитский А.А., Чусовитин А.Р.Поточный шифр на основе шифра Шеннона / XVI Российская конференция "Распределенные информационно-вычислительные рессурсы. Наука - цифровой экономике" (DICR-2017). 4-7 декабря 2017, ИВТ СО РАН, г. Новосибирск. (РИНЦ)
8) Е.В. Игоничкина Статистический анализ поточных шифров //https://cyberleninka.ru/article/v/statisticheskiy-analiz-potochnyh-shifrov
9) Diffie W., Hellman M.E.: Privacy and authentication: an introduction to cryptography. Proc. IEEE 67(3), 397–427 (1979).

10. Приложение А. Протокол шифрования алгоритмом Шеннона

```
from random import seed, randint, choice
from os import listdir


class Shannon:
    def __init__(self, count_files_key=4):
        # Количество файлов для шифрования
        self.count_files = 1024
        # Список файлов для шифрования
        self.files_code = list()
        # Список файлов отобранных для
        сессии self.files = set()
        # путь до директории text
        self.helper_dir = '..\\..\\helper'
        # Ключ шифрования
        self.key = ''
        # Количество ключей для шифрования
        self.count_files_key = count_files_key

    def _selection(self, private_key):
        """
        Метод для выбора файлов сессии и файлов для
        шифрования :param private_key:
        :return:
        """
        seed(private_key)
        self._selection_files()
        self._selection_files_code()

    def _selection_files(self):
        """
        Метод для формирования множества файлов для
        сессии :return:
        """
        file_list = listdir("%s\\text" % self.helper_dir)
        if len(file_list) < self.count_files:
            raise ('Ошибка', 'Нужно больше файлов для шифрования')
        self.files = set()
        while len(self.files) < self.count_files:
            self.files.add(choice(file_list))
        self.files = list(self.files)

    def _selection_files_code(self):
        """
        Метод выбирает файлы, которыми будет использовать для
        шифрования :param private_key:
        :return:
        """
        count_files = self.count_files_key
        self.files_code = set()
        while len(self.files_code) < count_files:
            self.files_code.add("%s\\text\\%s" % (self.helper_dir,
choice(self.files)))


    @staticmethod
    def _xor(text, code):
        """
```

```
        Метод выполняет операцию XOR к 2 последовательностям
        bytearray :param text:
        :param code:
        :return:
        """
        for index in range(len(text)):
            text[index] = text[index] ^ code[index % len(code)]
        return text

    def _make_code_key(self):
        """
        Метод формирует ключ шифрования
        :return:
        """
        for file in self.files_code:
            with open(file, 'rb') as f:
                code = bytearray(f.read())
                if self.key:
                    self.key, code = (code, self.key) if len(code) > len(self.key)
else (self.key, code)
                    self.key = self._xor(self.key, code)
                else:
                    self.key = code

    def _message_with_key(self, text):
        """
        Метод шифрует последовательность с ключом
        шифрования :param text:
        :return:
        """
        return self._xor(text, self.key)

    def _encode_message_with_files(self, text):
        """
        Метод шифрует последовательность рассчитывая ключ шифрования во
время шифрования
        :param text:
        :return:
        """
        for file_code in self.files_code:
            with open(file_code, 'rd') as f:
                code = f.read(1024)
        return text

    def _code_text(self, text):
        if isinstance(text, bytes):
            text = bytearray(text)
        return self._message_with_key(text)

    @staticmethod
    def _open_file(file, mode='rb'):
        if isinstance(file, str):
            file = open(file, mode)
        return file

    def _code_file(self, file_in, file_out=None):
        file_in = self._open_file(file_in)
        file_out = self._open_file(file_out, 'wb') if file_out else None
        while True:
            text = file_in.read(1024)
            if text == b'':
                break
            text = self._code_text(text)
```

```
            if file_out:
                file_out.write(text)

    def _switch(self, text, file_in=None, file_out=None):
        text_out = None
        if file_in and not file_out:
            text_out = self._code_file(file_in), self.files, self.files_code
        elif file_in and file_out:
            text_out = self._code_file(file_in, file_out), self.files,
self.files_code
        elif text:
            text_out = self._code_text(text), self.files, self.files_code
        return text_out

    def encode(self, text='', file_in=None, file_out=None):
        """
        Метод шифрует строку
        :param text:
        :param file_in:
        :param file_out:
        :return:
        """
        self._selection(randint(2**512, 2**1024))
        self._make_code_key()
        return self._switch(text, file_in, file_out)

    def decode(self, text, file_in=None, file_out=None):
        """
        Метод расшифровывает строку
        :param text:
        :param file_in:
        :param file_out:
        :return:
        """
        return self._switch(text, file_in, file_out)
```