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

ИССЛЕДОВАНИЕ И АНАЛИЗ НАДЁЖНОСТИ ШИФРА ШЕННОНА

Магистерская диссертация

по направлению 09.04.01 «Информатика и вычислительная техника»

Студент Анискина А.Д. /...../

Руководитель Ракитский А.А. /...../

Новосибирск 2019 г.

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

КАФЕДРА

ПРИКЛАДНОЙ МАТЕМАТИКИ И КИБЕРНЕТИКИ

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРАНТА

СТУДЕНТА Анискина А.Д. ГРУППЫ МГ-172

УТВЕРЖДАЮ

«___» _____ 2019 г.

Зав. кафедрой

_____/ А.Н.Фионов /

Новосибирск 2019 г.

1. Тема выпускной квалификационной работы магистранта

Исследование и анализ надёжности шифра Шеннона.
утверждена приказом СибГУТИ от «_24_» _мая_2019 г. № 4/1208о-1-17____

2.Срок сдачи студентом законченной работы « 25 » июня 2019 г.

3.Исходные данные к работе

1. 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.
2. 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 (дата обращения: 21.10.2018)
3. Ковтун В.Ю. Криптоанализ симметричных криптосистем: поточные шифры. Криптоанализ по побочным каналам NRJETIX, 2000 - 2008.- 6 с.

Содержание пояснительной записки (перечень подлежащих разработке вопросов)	Сроки выполнения по разделам
ВВЕДЕНИЕ	10.12.18 – 28.12.18
Изучение и анализ шифра Шеннона	05.01.19 – 07.02.19
Исследование криптографических атак на поточные шифры	15.02.19 – 10.03.19
Разработка модулей тестирования шифра Шеннона	11.03.19 – 15.04.19
Сбор, исследование и анализ результатов применения атак к шифру Шеннона	20.04.19 – 25.05.19
ЗАКЛЮЧЕНИЕ	26.05.19 – 20.06.19

Дата выдачи задания « 15 » декабря 2018г.
Руководитель _____
подпись

Задание принял к исполнению « 15 » декабря 2018 г.
Студент_____
_____*подпись*

АННОТАЦИЯ

Выпускная квалификационная работа Анискиной Алисы Денисовны
(Фамилия,И.О.)
по теме «Исследование и анализ надежности шифра Шеннона»

Объём работы - 40 страниц, на которых размещены 8 рисунков. При написании работы использовалось 12 источников.

Ключевые слова: криптография, поточный шифр, алгоритм Шеннона, статистический анализ, криптоанализ.

Работа выполнена на кафедре ПМиК СибГУТИ
(название предприятия, подразделения)

Цель работы:

Провести криптографический анализ реализации поточного шифра на основе алгоритма Шеннона. Тестирование реализации шифрования при помощи статистических тестов, исследование шифра на возможность проведения силовых, корреляционных атак, а также атак методом компромисс время-память.

Проанализирована теоретическая и практическая надежность шифрования, реализованы попытки раскрытия шифрующей гаммы, выявлены некоторые особенности алгоритма шифрования, сделаны выводы о криптографической стойкости алгоритма.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность таких проблем, как обеспечение безопасности информации, передаваемой по сети в телекоммуникационной среде, с каждым годом увеличивается всё больше. Наиболее эффективный способ защитить информацию от посторонних лиц — это шифрование, с помощью которого можно надежно скрыть данные, даже в случае их перехвата злоумышленниками. Однако, шифрование – это достаточно трудоемкий процесс, который может значительно уменьшить скорость передачи защищенной информации, а также требовательный к вычислительным ресурсам системы. К системам связи, предназначенным для обмена информацией, как со спутниками, так и с наземными объектами, предъявляются требования не только высокой защищенности передаваемой информации, но и скорости ее передачи. Это требует тщательного подхода к выбору алгоритмов шифрования данных. Для того, чтобы уменьшить потери производительности, а также ускорить шифрование данных, в настоящее время используются поточные шифры. Актуальность применения методов поточного шифрования, как в коммерческих приложениях, так и в открытых системах, объясняется весьма значительным ростом объемов трафика передачи данных в сетях связи. Американский криптоаналитик, основатель теории информации, Клод Шеннон в 1949 году предложил варианты алгоритмов поточного шифрования на основе криптографической системы Вернама. Он описал два алгоритма, которые совмещали бы в себе высокую защищенность данных, а также большую скорость шифрования, но не доказал данные свойства. Намного позже была теоретически доказана криптографическая стойкость предложенных алгоритмов, но не была проверена на практике.

Целью моей работы является практический анализ надежности одного из шифров, базирующемся на алгоритме Шеннона. В работе приводится анализ, статистические исследования шифра, и разбор атак, которые возможно применить к данному шифру.

1. Шифрование

1.1. Основные понятия и определения

Шифрование — это основной криптографический метод защиты информации, использующийся в моментах, когда требуется повышенный уровень защиты данных, обеспечивающий ее конфиденциальность. [1] Шифрованием информации является её обратимое преобразование с целью сокрытия данных для всех, за исключением авторизованных лиц, имеющих право доступа к шифруемой информации. Алгоритм, необходимый для осуществления шифрования и дешифрования, называется криптографическим алгоритмом.

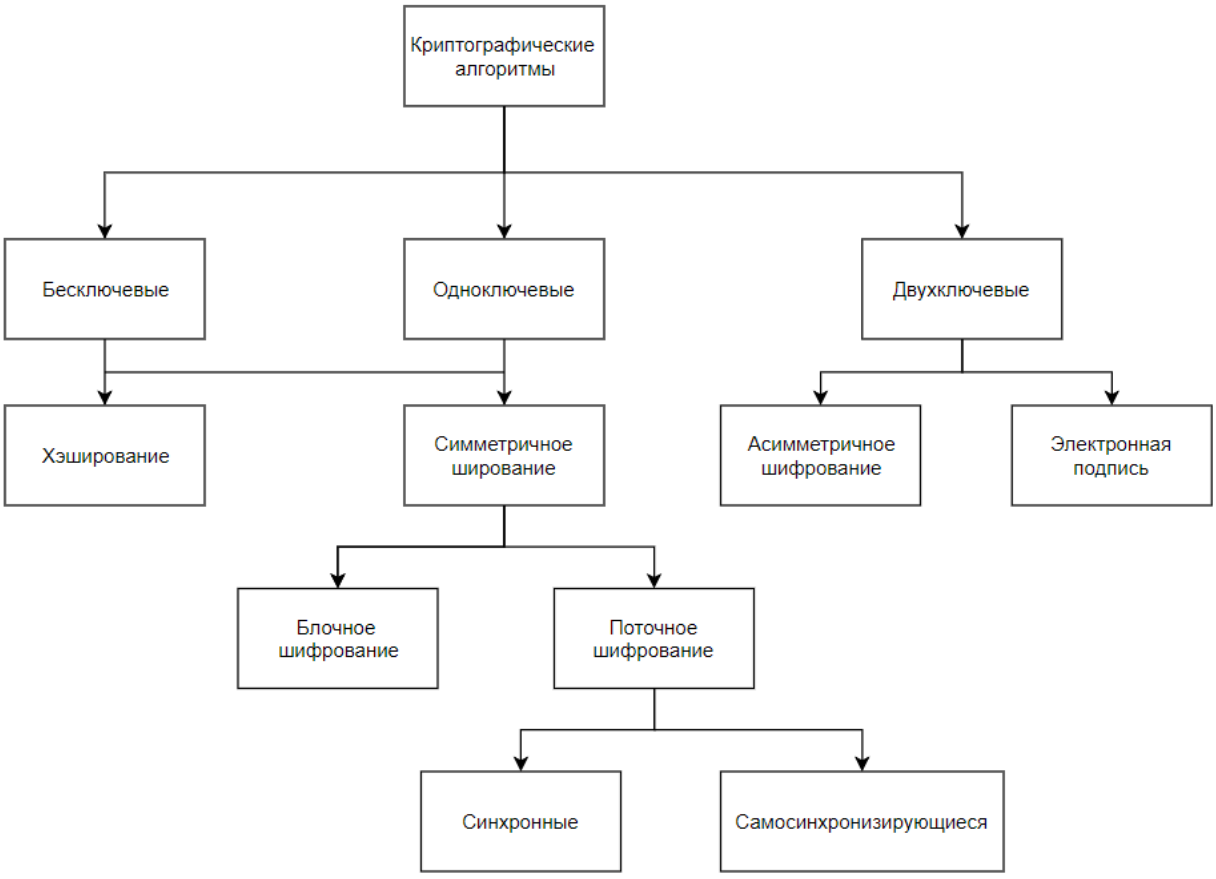


Рисунок 1.1 – Классификация криптографических алгоритмов

Криптоалгоритмы, в соответствии с рисунком 1.1, классифицируются по трём основным категориям:

a. Бесключевой алгоритм – не использующий в процессе преобразований никаких ключей.
b. Одноключевой алгоритм – использующий в своих вычислениях дополнительный параметр – секретный ключ.
c. Двухключевой алгоритм – использующий на различных этапах вычислений различные виды ключей (секретный и открытый).

К основным типам криптографических алгоритмов можно отнести такие, как:

a) Хэш-функции – метод криптографической защиты, выполняющий преобразование данных переменной длины в последовательность фиксированного размера.
b) Алгоритмы симметричного шифрования – алгоритм шифрования, при котором для шифрования и дешифрования информации используется один и тот же ключ. Симметричное шифрование, в свою очередь, делится на два вида:
    a. Блочное шифрование – алгоритм характеризуется предварительным разбитием информации на блоки фиксированной длины, после чего блоки поочередно шифруются.
    b. Поточное шифрование – в основном, применяется в тех случаях, когда информацию невозможно разбить на блоки. Алгоритмы потокового шифрования шифруют данные побитно либо посимвольно. Поточные шифры бывают синхронными – когда поток ключей генерируется независимо от открытого текста и шифротекста. При этом, потеря знака шифротекста приводит к сбою в синхронизации между генераторами и проблемам с расшифровкой остальной части текста. И самосинхронизирующимися – когда ключевой поток создаётся

функцией ключа и фиксированного числа знаков шифротекста. Каждое сообщение, передаваемое с использованием самосинхронизирующегося шифра, начинается со случайного заголовка с конкретным числом битов.

c) Алгоритмы асимметричного шифрования – алгоритмы шифрования, применяющие на различных этапах преобразований два различных ключа: открытый ключ для шифрования информации и секретный для дешифрования. Оба ключа связаны между собой соотношением, позволяющим легко и быстро вычислить открытый ключ, имея секретный, но в то же время свести к нулю возможность вычисления секретного ключа из открытого, при наличии ограниченного количества времени и ресурсов.

d) Алгоритмы электронной подписи – применяется для подтверждения целостности и авторства данных, в алгоритме также используется два типа ключей: секретный ключ - для вычисления электронной подписи, а, вычисляемый из секретного, открытый ключ – для проверки подписи.

Целью моей дипломной работы является проверка надежности шифра, основанном на алгоритме Шеннона. Рассматриваемый шифр относится к алгоритмам симметричного синхронного поточного шифрования. В связи с этим, в дальнейшем подробно будут рассмотрены только эти категории шифрования.

1.2. Поточные алгоритмы шифрования

Поточный шифр — это симметричный шифр, в котором каждый символ открытого текста преобразуется в символ шифрованного текста в зависимости не только от используемого ключа, но и от его расположения в потоке открытого текста. Генератор ключей выдает поток битов k_i, которые будут использоваться в качестве гаммы, источник сообщений генерирует биты, являющиеся открытым текстом x_i. Шифрование происходит путем логических операций над битами ключа

и битами открытого текста, в результате чего получаются биты зашифрованного сообщения y_i.

Чтобы из шифротекста y_1, y_2,..., y_n восстановить сообщение x_1, x_2,..., x_n, необходимо сгенерировать точно такую же ключевую последовательность k_1, y_k,..., k_n, что и при шифровании, и использовать для расшифрования обратную формулу.

Важнейшим достоинством поточных шифров перед блочными является высокая скорость шифрования, соизмеримая со скоростью поступления входной информации; поэтому, обеспечивается шифрование практически в реальном масштабе времени вне зависимости от объема и разрядности потока преобразуемых данных.

Поточный алгоритм шифрования устраняет необходимость разбивать сообщение на целое число блоков достаточно большой длины, следовательно, он может работать в реальном времени. Таким образом, если передается поток символов, каждый символ может шифроваться и передаваться сразу. При этом шифрование и дешифрование может обрываться в произвольный момент времени. И как только связь восстановлена можно продолжать процедуру без проблем. Работа типичного поточного шифра представлена на рис. 1.2.

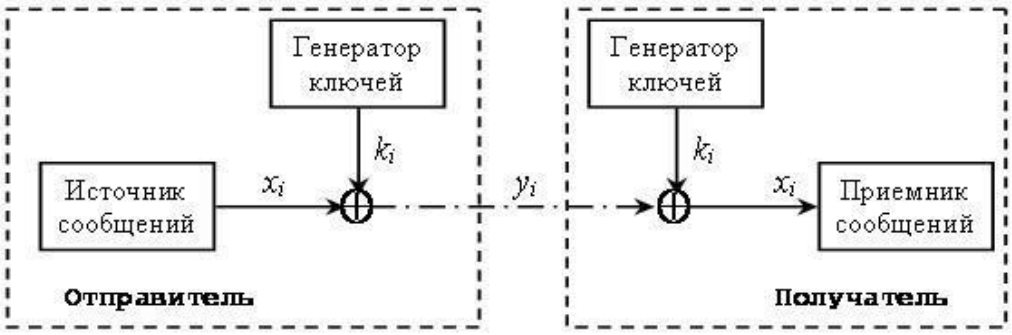


Рисунок 1.2. – Схема работы поточного шифра

Как правило, открытый текст и гаммирующая последовательность являются независимыми битовыми потоками. Следовательно, так как шифрующее и дешифрующее преобразование для всех поточных шифров аналогично, они

должны быть различны только способом построения генераторов гаммирующей последовательности. Таким образом, защищенность системы напрямую зависит от свойств генератора потока ключей. Чтобы избежать взлома, при создании поточного шифра необходимо соблюдать два критерия, обеспечивающих надежность:

- гамма должна обладать хорошими статистическими характеристиками
- период гаммы должен превышать длину самого большого сообщения, которое может быть передано и зашифровано с использованием одного ключа

Если гамма последовательность не имеет периода и выбирается случайным образом, то, теоретически, и взломать шифр невозможно. [2] Однако, в связи с тем, что в существующих криптографических системах не встречаются закрытые ключи, имеющие с открытым текстом одинаковую длину, на практике применяются ключи меньшей длины. При помощи таких ключей можно генерировать лишь псевдослучайные последовательности, соответственно, такое свойство гамма последовательности в будущем может быть выявлено и использовано при совершении атаки на поточный шифр.

1.3 Структура алгоритмов симметричного шифрования.

Особенностью симметричного шифрования является использование единого ключа для шифрования и дешифрования информации. В стандарте, определяющим алгоритм симметричного шифрования, понятия ключа определено, как: «Конкретное секретное состояние некоторых параметров алгоритма криптографического преобразования, обеспечивающее выбор одного преобразования из совокупности всевозможных для данного алгоритма преобразований». [3]

Симметричное шифрование и на текущее время остаётся наиболее актуальным и надёжным методом защиты информации. Алгоритмы имеют высокую

производительность, криптография с единым ключом стойкая, что делает процесс дешифрования без знания ключа практически не выполнимой задачей. К достоинствам симметричных систем шифрования можно отнести высокую скорость шифрования, простоту реализации и большую изученность, относительно асимметричных алгоритмов. Несмотря на достаточное количество положительных моментов, симметричные алгоритмы имеют существенный недостаток в виде проблемы распределения ключей. В связи с тем, что в шифровании и дешифровании используется один секретный ключ, который должен быть известным как отправителю, так и получателю, требуются высоко надежные механизмы для передачи ключей. Большинство симметричных шифров имеют подобный алгоритм обработки информации.

1.4 Синхронные алгоритмы шифрования

Синхронными поточными шифрами называются шифры, в которых поток ключей генерируется независимо от открытого текста и шифротекста.

В процессе шифровании с использованием синхронного алгоритма, генератор потока ключей выдаёт биты потока ключей, которые идентичны битам потока ключей при дешифровании. Основная сложность в данном подходе заключается в необходимости синхронизации генераторов ключа на передающей и приемной сторонах. Если в процессе передачи произошла Потеря знака шифротекста или вставка хотя бы одного бита, то вся последовательность битов шифрованного текста после ошибочного бита не сможет быть расшифрована приведёт к нарушению синхронизации между этими двумя генераторами и невозможности расшифрования оставшейся части сообщения. В данной ситуации отправитель и получатель должны повторно синхронизоваться для продолжения работы. Синхронизация должна выполняться так, чтобы ни одна часть потока ключей не была повторена, поэтому переводить генератор в более раннее состояние не имеет смысла.

Как правило, синхронизация производится вставкой в передаваемое сообщение специальных маркеров. В результате этого, пропущенный при передаче знак приводит к неверному расшифрованию до тех пор, пока не будет принят один из маркеров.

Среди преимуществ синхронных поточных шифров выделяют следующие свойства:

- Отсутствие эффекта распространения ошибок - один искаженный бит при передаче приведет к искажению только одного ошибочного бита текста при расшифровывании. Однако, они не защищают полностью от битовых сбоев.
- Синхронные шифры защищают от возможных вставок и удаления отрезков шифрованного текста из потока. Такие операции приведут к нарушению синхронизации, что будет сразу же обнаружено на приемной стороне.

В качестве недостатков синхронных шифров выделяют возможность изменения определенных бит шифротекста. При наличии на руках открытого текста злоумышленник может скорректировать биты так, как ему необходимо.

Таким образом, характерными особенностями поточного шифрования являются высокая производительность, отсутствие эффекта размножения ошибок, малая уязвимость, благодаря чему оно применяется при шифровании информации в каналах связи.

Лишь поточные шифры обеспечивают приемлемую скорость шифрования на канальном, сетевом и транспортных уровнях, что обеспечивает их применимость в стандартах GSM, GPRS, UMTS и т. д.

1.5 Область применения, существующие сейчас шифры

В отличие от блочного шифрования, поточное шифрует поток данных поэлементно, без задержки с высокой скоростью преобразования, которая соизмерима со скоростью поступления открытого текста. В связи с этим можно сделать вывод, то при поточном шифровании входной информации, вне зависимости от разрядности потока преобразуемых данных и его объема, обеспечивается шифрование в реальном масштабе времени. Исходя из данных статьи [4] можно выделить основные области для эффективного применения поточных шифров:

- Сверхвысокоскоростное шифрование при скорости входящего потока данных превышающей несколько Гб/с.
- Системы с ограничениями по параметрам аппаратно-программных средств криптографической защиты. Примером могут являться портативные и мобильные устройства, специальная аппаратура с ограниченным объемом памяти или потребляемой мощностью.
- Ранее описанные области, объединенные со особенными требованиями, предъявляемыми к свойствам алгоритмов, которые эффективно реализовать можно только поточными методами. Примером могут быть поточная цифровая подпись или проверка целостности в режиме реального времени.

В настоящее время широкое применение получили такие алгоритмы синхронных поточных шифров, как A5, RC4, Chameleon, SOBER, Leviathan.

- A5 - поточный алгоритм шифрования, который по настоящее время используется для обеспечения защищенности передаваемых данных между телефоном и базовой станцией в европейской системе мобильной цифровой связи GSM.
- RC4 – поточный алгоритм шифрования с переменной длиной ключа, был разработан Роном Ривестом в 1987 г. для компании RSA Data Security, Inc.

В настоящее время шифр применяется в различных системах защиты информации в компьютерных сетях, таких как протоколы TLS, SSL, программа сертификации устройств беспроводной связи WEP и WPA. Шифр реализован в таких коммерческих криптопродуктах, как, например, Lotus Notes, Apple Computers AOCE, Oracle Secure SQL.

- Chameleon – поточный алгоритм шифрования, разработанный Р. Андерсоном. Яркими особенностями шифра являются высокая криптостойкость и свойство, благодаря которому незначительные изменения в ключе вызывают такие же незначительные изменения в гамме. Описанное свойство играет большую роль в случае применении шифра в ряде коммерческих приложений, например, при защите интеллектуальной собственности в платном телевидении. Схема шифра позволяет обнаруживать нарушителей в попытках незаконно тиражировать информацию, предназначенную для индивидуального пользования.
- SOBER – поточный алгоритм шифрования, использующий различные методы для повышения скорости формирования гаммирующей последовательности. Шифр был разработан специально для применений с мобильными телефонами CDMA.
- Leviathan – поточный алгоритм шифрования, который был разработан в компании Cisco Systems и ориентирован в основном на сетевые приложения. Процесс шифрования заключается в побитовом сложении соответствующих элементов гаммы и исходного сообщения. Поток ключевых элементов определен набором двоичных деревьев, что позволяет реализовать возможность поиска без использования линейной функции обновления состояния и без использования специальной функции поиска.

2 Криптоанализ

2.3 Общие понятия криптоанализа

Основной и самой важной целью криптоалгоритма является гарантированная защита информационных данных. Вследствие данного факта, главным свойством криптоалгоритма является криптографическая стойкость. Криптографической стойкостью называют способность алгоритма противостоять возможным атакам. Атакующие криптоалгоритм используют методы криптоанализа. Определение криптоанализа, под которым понимается научная и практическая деятельность по исследованию криптографических алгоритмов с целью получения обоснованных оценок их криптографической стойкости, было введено в 1920 году Уильямом Фридманом: «Стойким считается алгоритм, который для своего вскрытия требует от противника практически недостижимых вычислительных ресурсов или недостижимого объема перехваченных зашифрованных сообщений или времени раскрытия, которое превышает время жизни интересующей противника информации». [5] В большинстве случаев невозможно математически доказать криптостойкость алгоритма, а можно доказать только уязвимости.

2.4 Криптоанализ поточных шифров

Структура поточного ключа может иметь некоторые уязвимые места, которые позволят нападающему получить некоторую дополнительную информацию о ключе. Наиболее очевидно, если период поточного ключа (т.е. количество бит, после которых поточный ключ начинает повторяться) слишком мал, то нападающий может применить обнаруженные части поточного ключа для дешифрации других частей закрытого текста.

Поскольку при разработке любых схем преобразования их стойкость определяется прежде всего стойкостью к известным на текущий момент аналитическим атакам, направленным на выявление в рассматриваемой схеме слабостей различного рода, представляется целесообразным рассмотрение

наиболее распространенных и упоминаемых методов анализа для разработки требований к схемам поточного преобразования информации. [7]

При рассмотрении методов анализа схем поточного преобразования все методы можно условно разделить на три класса:

- аналитические атаки;
- статистические;
- силовые.

К аналитическим атакам относят атаки, в которых алгоритм построения атаки основан на аналитических принципах вскрытия схемы. К статистическим атакам относятся атаки, основанные на оценке статистических свойств управляющей последовательности. К силовым атакам относятся атаки, основанные на принципе полного перебора всех возможных комбинаций ключа; теоретически, при попытке вскрытия схемы преобразования данный вид атаки должен быть наиболее эффективным по сравнению с остальными предлагаемыми видами атак.

Также установлен набор требований, которым должны удовлетворять стойкие криптосистемы на основе поточных шифров. В идейном плане методы взлома поточных шифров сводятся к одному из следующих двух подходов:

- использование статистических связей (корреляционные атаки);
- линеаризация (сведение задачи поиска ключа к решению системы линейных уравнений).

Соответственно, от стойких поточных схем требуется:

- большие периоды выходных последовательностей;
- хорошие статистические свойства выходных последовательностей;
- высокая линейная сложность выходных последовательностей.

2.5 Основные атаки на поточные шифры

Силовые атаки, или так называемые атаки методом грубой силы, базируются на принципе полного перебора всевозможных вероятных комбинаций ключа. Если вычислительная сложность криптоаналитической атаки меньше вычислительной сложности полного перебора всевозможных ключевых комбинаций, то считается, что атака была успешной.

При тестировании метода грубой силы и известных открытых и зашифрованных текстах, входное сообщение зашифровывается и полученный результат сравнивается с имеющимся шифртекстом. Если результат шифрования и шифртекст совпали, значит искомый ключ был подобран верно.

Если же из имеющихся у криптоаналитика данных есть только шифртекст, то ситуация с тестированием метода усложняется. При таких исходных данных требуется наличие какой-либо информации о входящем сообщении [3], например:

- Если входящее сообщение представляет собой осмысленный текст на каком-либо языке, то шифртекст должен иметь достаточный размер (минимальный такой размер называется точкой единственности) для однозначного дешифрования в осмысленный текст. Для английского языка точка единственности теоретически определена, как 27 букв, если же сообщение будет короче, то при полном переборе есть вероятность появления нескольких различных осмысленных текстов. В таком случае, если возможность перехватить несколько шифртекстов отсутствует, то становится невозможным определить, какое из найденных сообщений является исходным.
- Если открытый текст является бинарным, то необходима информация о структуре текста. Например, если перехватываемые данные представляют собой заархивированный файл, то при переборе ключей каждое значение должно рассматриваться, как возможный заголовок архива.

Несмотря на простоту метода, существуют различные методы его улучшения, в частности, распараллеливание работы.

Аналитическими атаками называются атаки, в которых принцип построения атаки базируется на аналитических принципах взлома криптографического алгоритма. Класс аналитических атак можно разбить на два подкласса [8]:

- методы криптоанализа шифрующей гаммы
- методы криптоанализа процедуры ключевой инициализации и реинициализации

При методах криптоанализа шифрующей гаммы, в связи с особенностями построения поточных шифров, одним из наиболее используемых видов атак на данные схемы является корреляционная атака.

Известно, что работа по вскрытию криптосистемы может быть значительно сокращена, если нелинейная функция пропускает на выход информацию о внутренних компонентах генератора. Основная идея корреляционных атак состоит в нахождении корреляции между шифрующей гаммой и различными линейными комбинациями ключа. Объектом исследования в корреляционных атаках выступают нелинейные функции, вносящие нелинейность в выходную последовательность регистра сдвига, соответственно, в зависимости от устройства применяемой нелинейной функции, реализации корреляционных атак постоянно будут различны и будут основаны на особенностях устройства анализируемой функции [9].

Также, одной из основных атак разряда аналитических относится атака компромисс время-память, целью которой является восстановление исходного состояния регистра сдвига, используя известную схема устройства и фрагмент шифрующей последовательности. В качестве примера, такая атака была применена к поточному шифру A5 [12]. Алгоритм атаки содержит в себе два этапа:

1. построение словаря, включающего в себя всевозможные пары «состояние-выход» размерности *n*;
2. предположение об определенном начальном заполнении ячеек памяти.

На основе входных данных генерируется выход, после чего просматривается выходная последовательность и ищется соответствие со сгенерированным выходом. Если произошло совпадение, то данное предположительное заполнение с большой вероятностью является начальным.

При использовании данных атак предполагается, что криптоаналитик имеет знание об описании генератора, а также об открытом и соответствующем ему закрытом текстах. Задачей криптоаналитика является определение применяемого ключа (начального заполнения). На рис. 2.1 представлены наиболее известные криптоаналитические атаки, применяемые к синхронным поточным шифрам.

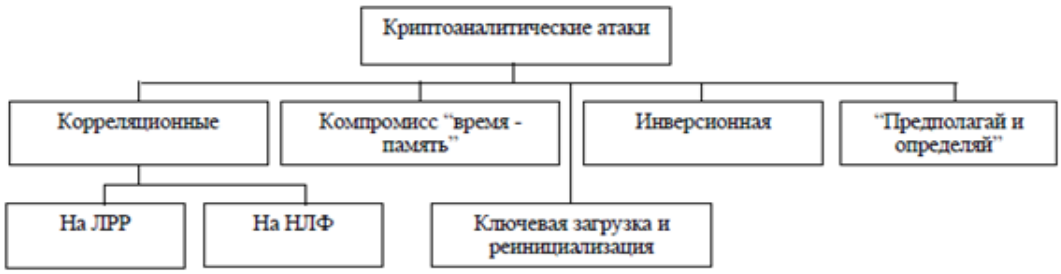


Рисунок 2.1 - Классификация криптоаналитических атак

Статистические атаки базируются на оценке статистических свойств шифрующей гаммы. Этот класс атак делится на два подкласса:

- методы криптоанализа сложности последовательности
- методы криптоанализа свойств шифрующей гаммы

Методы криптоанализа сложности последовательности направлены на то, чтобы выявить возможность генерировать последовательности, которая была бы аналогична шифрующей гамме, каким-либо другим способом, сложность реализации которого была бы сравнительно меньше способа генерации исходной

шифрующей гаммы. В идеале, найденный способ генерации возможно было бы применить на практике.

Методы криптоанализа свойств шифрующей гаммы направлены на то, чтобы выявить возможный дисбаланс выходной последовательности криптографического алгоритма с целью найти способ предположить следующий бит выходной последовательности с большей вероятностью, чем при его случайном выборе. Описанные методы оперируют различными статистическими тестами, выбор необходимого и достаточного количества тестов – право криптоаналитика.

Управляющая гамма должна иметь длину периода многократно превосходящую длину открытого текста и не содержать длинных повторяющихся отрезков. В противном случае криптоанализ значительно упрощается.

2.6 Статистические тесты

Надежный шифр должен быть устойчивым к криптоанализу, а значит обладать свойствами перемешивания и рассеивания, то есть каждый бит исходных данных должен оказывать влияние на все биты шифртекста, а характер этого влияния не должен подвергаться как статистическому, так и к алгоритмическому анализу. Это означает отсутствие легко обнаружимых статистических связей между открытым и зашифрованным текстом, а зависимости между ними должны быть достаточно сложными, настолько, чтобы обнаружить их путем анализа было практически невозможно.

Для проверки соответствия криптографического алгоритма данному требованию используют статистические тесты. Если при помощи тестирования удается доказать, что распределение не является равномерным, в последовательностях данных имеют место статистические недостатки шифра (в англоязычной литературе поиск таких недостатков называется “distinguishing attack”).

Для того, чтобы обеспечить абсолютную стойкость алгоритму количество секретной информации в его ключе должно быть максимально возможным для размера ключа, что означает равновероятность и статистическую независимость всех битов ключевого элемента. Есть конкретная обозначенная граница между достаточно и недостаточно хорошо спроектированными шифрами в отношении статистических связей между входными и выходными данными: хорошо спроектированный шифр невозможно раскрыть способом, который более эффективен, чем полный перебор всех возможных значений ключа. Для раскрытия недостаточно хорошо спроектированного шифра могут быть использованы и более эффективные способы. Если результаты статистического теста для определенного шифра не соответствуют требуемым условиям, то такой шифр не рекомендуют к применению.

2.7 Стопка книг

Тест «Стопка книг» (в англоязычной литературе “move-to-front”) предназначен для проверки гипотезы о том, что элементы выборки $Z = \{z1, z2, \dots, zn\}$ из алфавита $A = \{a_1, a_2, \dots, a_n\}$ имеют равномерное распределение, то есть они независимы и $P(z_n = a_i) = 1/S$, $n = 1 \dots n$, $i = 1 \dots S$

Тестирование описываемым методом происходит следующим образом: в начале тестирования выборки, порядок букв алфавита A (возможно, произвольный) фиксируется, а сами буквы получают нумерацию, соответственно порядку от 1 до S. Множество номеров разбиваются на непересекающиеся части $A1 = \{a_1, a_2, \dots, a_k\}$, $A2 = \{a_{k+1}, a_{k+2}, \dots, a_S\}$. Установленный порядок букв подвергается изменениям после каждого анализа значения выборочного элемента z_i следующим образом: элемент ziвстает в начало выборки и получает номер 1, элементы, номера которых были меньше, чем номер z_i, увеличиваются на 1, нумерация остальных элементов остается неизменной.

После анализа выборки производится подсчет v_n - количества номеров, принадлежащих части *A1*, то есть номеров, к которым обращение осуществлялось наиболее часто. Соответственно, *N*-v_n – количество номеров, принадлежащих части *A2*, с более редким количеством обращений. В завершение, вычисляется статистика: $x^2 = \frac{(v_n - NP_1)^2}{NP_1} + \frac{(N - v_n) - N(1 - P_1)^2}{N(1 - P_1)}$, $P_1 = \frac{|A1|}{S}$ Если x^2 меньше критического уровня $\chi^2_{1,1-\alpha}$, то гипотеза о равномерности распределения элементов выборки принимается, иначе - отвергается. [6]

Основной идеей данного метода является подсчет встречаемости номеров букв, а не частоты встречаемости букв в выборке. Если выполнена гипотеза, отрицающая равновероятное распределение элементов выборки, вероятность и частота встречаемости некоторых букв будет больше 1/S, и их номера, в среднем будут меньше, чем у букв с меньшими вероятностями. Если выполнена противоположная гипотеза, говорящая о равновероятном распределении, то вероятность появления в выборке элемента с любым номером равно 1/S.

3 Шифр на основе алгоритма Шеннона

Среди поточных шифров выделяют такой подвид, как шифры с бегущим ключом - их особенностью является то, что открытый текст, ключи шифрования и зашифрованный текст являются последовательностями символов одного языка. Примером шифра с бегущим ключом может служить криптосистема Вернама, в которой ключевая последовательность представлена не случайным набором символов, а осмысленным текстом.

В одной из своих работ [10] Шеннон рассматривал криптосистему Вернама и выявил некоторые недостатки и слабые места в защищенности данного шифра. В частности, избыточность текста, используемого в качестве ключевой последовательности, существенно снижала криптостойкость алгоритма и приводила к тому, что при взломе шифра достаточно легко происходило выявление статистических зависимостей последовательностей символов.

После анализа рассматриваемого алгоритма и его недостатков, Шеннон предположил, как именно можно было бы усовершенствовать алгоритм шифрования Вернама, чтобы повысить надежность и безопасность шифра. Он предложил два алгоритмических варианта улучшения ключевой последовательности, используемой для шифрования.

В первом варианте рассматривалась теория, в которой вместо одного текста на английском языке используются несколько различных текстов, и все они суммируются с открытым текстом при помощи логической операции сложение по модулю два. При увеличении количества текстов, используемых при шифровании данных, криптостойкость алгоритма будет возрастать.

Во втором варианте усовершенствования была предложена идея использовать только определенные буквы текста, пропуская все остальные символы, что позволи тем самым сделать выбор каждого следующего символа

практически независимым. Алгоритм Вернама с данной доработкой, также должен был дать достаточно защищенный шифр.

Но Шеннон не привел никаких доказательств надежности доработанных алгоритмов и не исследовал их свойства, а только описал возможные улучшения. Намного позже, Рябко Б.Я. и Ракитский А.А. в своей работе [11] привели доказательство теоретической надежности обоих алгоритмов шифрования, описанных Шенноном. Исходя из представленных исследований, при использовании первого алгоритма Шеннона зашифрованный текст не может быть расшифовран в отсутствии ключевой последовательности, если шифрование открытого текста производилось при помощи более чем четырех различных текстов. Зашифрованный текст полученный при помощи второго алгоритма шифрования невозможно дешифровать в отсутствии ключа.

В данной дипломной работе рассматривается алгоритм и реализация только одного из предложенных улучшений, а именно метод, содержащий сложение некоторого количества текстов по модулю два.

3.3 Описание алгоритма шифрования

Есть два абонента А и Б, которые хотят обменяться друг с другом сообщениями так, чтобы их содержимое никто не узнал. У двух абонентов есть одинаковый набор 1024 файлов, каждый из которых имеет размер от 1 до 8 Кбайт. Некоторые из этих файлов будут использованы абонентами при шифровании в качестве ключевого элемента. После того, как файлы выбраны, чтобы зашифровать сообщение необходимо к открытому тексту, представленному в двоичном виде и выбранным файлам, также представленным в двоичном виде, применить операцию побитового сложения по модулю два.

Если длина передаваемого сообщения оказывается больше, чем длины имеющихся у абонентов ключевых файлов, то возникает необходимость расширять имеющиеся ключевые файлы. Расширение производится при помощи

дублирования данных, то есть применения операции конкатенации файла с самим собой. Статистические тесты, выявляющие закономерности в шифртекстах могут выявить уязвимость в алгоритме, в котором используется цикл, поэтому есть одно требование, которое должно соблюдаться, чтобы не снижать надежность шифрования - ключевые файлы, выбранные для шифрования, должны иметь попарно взаимно-простые значения длины.

Данное требование объясняется тем, что при применении операции сложения к файлам, которые ранее были продублированы, образуется цикл, длина которого будет равна наименьшему общему кратному: *НОК(E_1…E_n)*. В случае же когда длины ключевых файлов являются взаимно простыми, данное значение будет равно: $\text{НОК}(E_1 \dots E_S) = |E_1| \times \dots \times |E_S|$

Описанное свойство позволяет значительно увеличить длину цикла, настолько, чтобы можно было избежать выявления статистических зависимостей.

3.4 Достоинства и недостатки алгоритма Шеннона

К достоинствам алгоритма можно отнести такие его свойства, как:

- Высокая криптографическая стойкость алгоритма, обусловленная тем, что гамма-последовательность, которой производится шифрование открытого текста, не является результатом работы генератора, следовательно невозможно предсказать или алгоритмически просчитать возможные значения ключей шифрования.
- Алгоритм шифрования не содержит в себе сложных преобразований, а содержит только логическую операцию сложения по модулю два между бинарными последовательностями, которую можно проводить на векторах, что обеспечивает высокую скорость шифрования.

К недостаткам алгоритма относится:

- Необходимость абонентам хранить некоторое количество файлов, требуемых для шифрования. Для хранения 1024 файлов размером от 1 до 4 КБайт абоненту потребуется выделить на ключевые файлы около 4 Мбайт памяти. Также, сессионные файлы время от времени необходимо будет обновлять, что также требует загрузки на устройство файлов размером около 4 Мбайт.
- Еще одним недостатком является передача номеров сессионных файлов, которые будут использоваться как ключевые элементы при непосредственном шифровании открытого текста. Передача этих номеров должна производиться секретно. Следовательно, передача номеров ключевых файлов между абонентами будет осуществлена при помощи другого шифра. Исходя из этого можно сделать вывод, что шифр на основе алгоритма Шеннона будет надежным ровно настолько, насколько защищён шифр, при помощи которого передаются номера ключевых файлов.

4. Реализация шифра Шеннона

Чтобы реализовать защищенный канал связи, использующий шифрование сообщений при помощи алгоритма Шеннона, между двумя абонентами, необходимо обеспечить абонентов ключевыми файлами. Каждому абоненту передается 1024 файла, которые являются последовательностями символов от 1 до 4 КБайт, полученные из двоичного перевода rar-архива с отбрасыванием заголовка.

После этого, каждый абонент выбирает 10 файлов, которые одинаковы для каждого из абонентов, и производит над ними логическую операцию сложение по модулю два. Для выбора секретных файлов, являющихся гаммирующей последовательностью, у двух абонентов имеется закрытый ключ, известный только им, параметризирующий настройку генератора псевдослучайных чисел. Сгенерированные в будущем псевдослучайные числа определяют 10 секретных файлов, использующихся при шифровании.

Далее, непосредственное общение между двумя абонентами происходит следующим образом:

Первый абонент шифрует сообщения при помощи 10 выбранных секретных файлов. Он складывает побитово по модулю два между собой все 10 файлов и свой открытый текст. После того, как сообщение зашифровано, шифртекст отправляется второму абоненту.

Второй абонент принял зашифрованную последовательность и начал расшифровку сообщения. Используя те же 10 файлов, что и первый абонент, он снова производит побитовое сложение по модулю два над 10 секретными файлами и шифртекстом, чтобы из зашифрованного сообщения получить открытый текст.

Схема реализации шифра на основе алгоритма Шеннона будет выглядеть следующим образом (см. рис. 4.1):

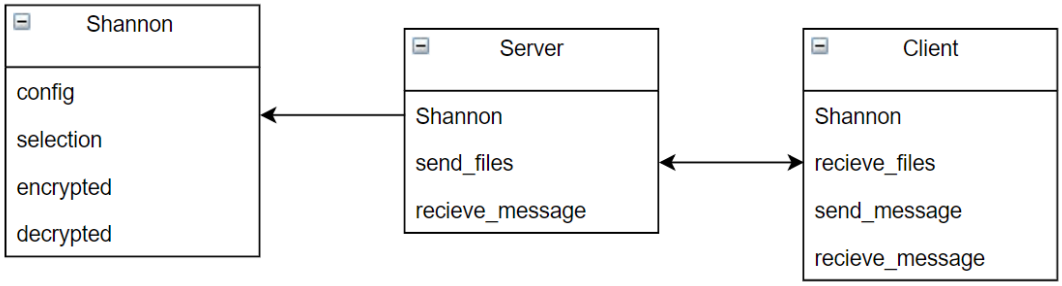


Рисунок 4.1 - UML -диаграмма шифра на основе алгоритма Шеннона

5. Исследование надежности шифра Шеннона

Для того, чтобы провести анализ надежности реализации шифра на основе алгоритма Шеннона, были проведены следующие исследования:

- статистическое исследование зашифрованных последовательностей
- исследование возможности проведения силовых атак:
    - статистическое исследование результатов применения силовой атаки при помощи одного файла
    - статистическое исследование результатов применения силовой атаки при помощи нескольких файлов, включая ключевые
- исследование возможности проведения корреляционных атак

Статистические исследования были проведены при помощи теста “Стопка книг”. Были написаны вспомогательные модули для обработки и исследования данных, полученных в результате проведенных атак. Для проведения исследований потребовались следующие предварительные шаги:

1. Были взяты 20 000 файлов, содержащих различные тексты на естественном языке.
2. Ключевые файлы были получены при помощи алгоритма сжатия PPMD следующим образом - файл с текстом был сжат в архив rar-типа, после чего во избежании обнаружения статистических зависимостей, из полученного файла архива был удален заголовочный фрагмент.
3. Каждый ключевой файл, полученный в ходе преобразований был проверен статистическим тестом. Если при тестировании были обнаружены зависимости между символами, то данный файл нельзя использовать в качестве ключевого элемента и он не будет входить в выборку.
4. Для одной сессии шифрования из набора ключевых файлов случайным образом было выбрано 1024 файла, являющихся сессионными файлами.

5. Для непосредственного шифрования открытого текста было выбрано определенное количество файлов, являющихся сессионными ключевым файлами.
6. В качестве открытого текста, к которому был применен шифр на основе алгоритма Шеннона, был взят файл формата .rar, длиной 16 МБайт.
7. Для получения более полных статистических данных, в каждом случае было проведено 1000 сессий шифрования (дешифрования).

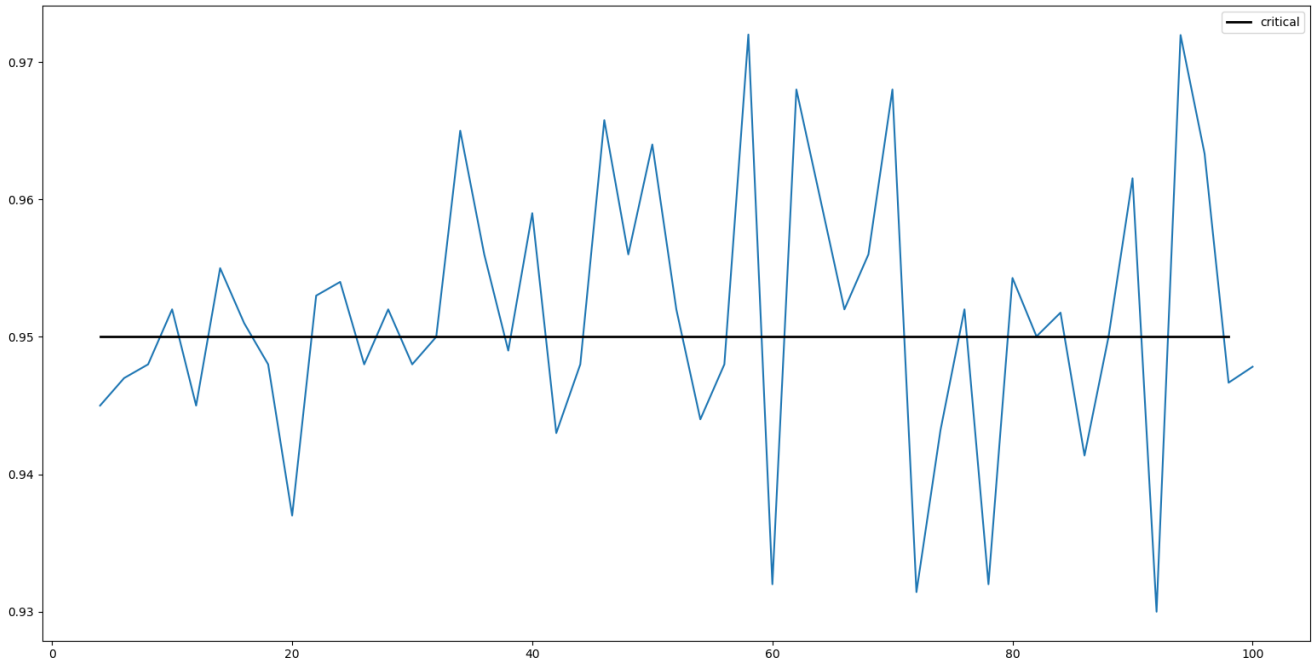
5.1 Статистическое исследование процесса шифрования

Для того, чтобы провести статистическое исследование зашифрованной последовательности, полученной после применения шифра, были проведены следующие действия:

1. Количество сессионных ключей, использующихся при непосредственном шифровании файла равно n, где $n = 1 \ldots 100$.
2. После того, как файл был зашифрован n-файлами, к полученной последовательности был применен статистический тест.
3. Результат прохождения теста интерпретируется следующим образом: пройденным считается тест, в которым не было обнаружено статистических зависимостей, проваленным тестом считается тот, в котором зависимости были обнаружены.

Было произведено 1000 сессий шифрования при помощи различных случайный сессионных файлов, для получения хороших статистических данных. Полученные значения были усреднены и отображены в графике зависимости количества пройденных статистических тестов от количества сессионных ключей, использующихся при шифровании (рис. 5.1).

В теоретическом доказательстве надежности шифра, описанном в статье [10], порог количества необходимых сессионных ключей равен 4 файлам.



5.1 - График зависимости количества пройденных статистических тестов от количества сессионных ключей

Исходя из данных графика видно, что процент файлов, успешно прошедших тест стопки книг, варьируется в пределах от 93% до 97%, что подтверждает хорошие статистические свойства шифртекстов.

6. Исследование применения метода силовых атак

При имеющихся данных N - количество файлов участвующих в формировании ключа, N принимает значения от 4 до S, трудоемкость взлома шифра на основе алгоритма Шеннона при помощи полного перебора ключей составит:

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

Таким образом при S = 20, $T \sim 2^{139}$, при S = 40, $T \sim 2^{240}$.

6.1 Применение метода силовой атаки при помощи одного файла

Далее будет проведен анализ того, насколько повлияет на статистические свойства перебор всевозможных сессионных файлов и применение каждого из них к зашифрованной последовательности. Для того, чтобы провести исследование результатов применения силовой атаки при помощи одного файла были проведены следующие действия:

1. Количество сессионных ключей, использующихся при непосредственном шифровании файла равно n, где $n = 4 \ldots 60$.
2. После того, как файл был зашифрован n-файлами, к полученной последовательности и одному из сессионных файлов применяется операция логического сложения по модулю два. Сессионные файлы перебираются по очереди по одному и могут являться как просто сессионным файлом, так и сессионным ключевым файлом, который ранее уже применялся к открытому тексту.
3. Для получения результата исследования к полученной последовательности применяется статистический тест, чтобы определить были ли снижены статистические характеристики последовательности.

Было произведено 1000 сессий шифрования при помощи различных случайный сессионных файлов, полученные значения были усреднены и

отображены в графике зависимости количества пройденных статистических тестов от количества сессионных ключей, использующихся при шифровании (рис. 6.1).

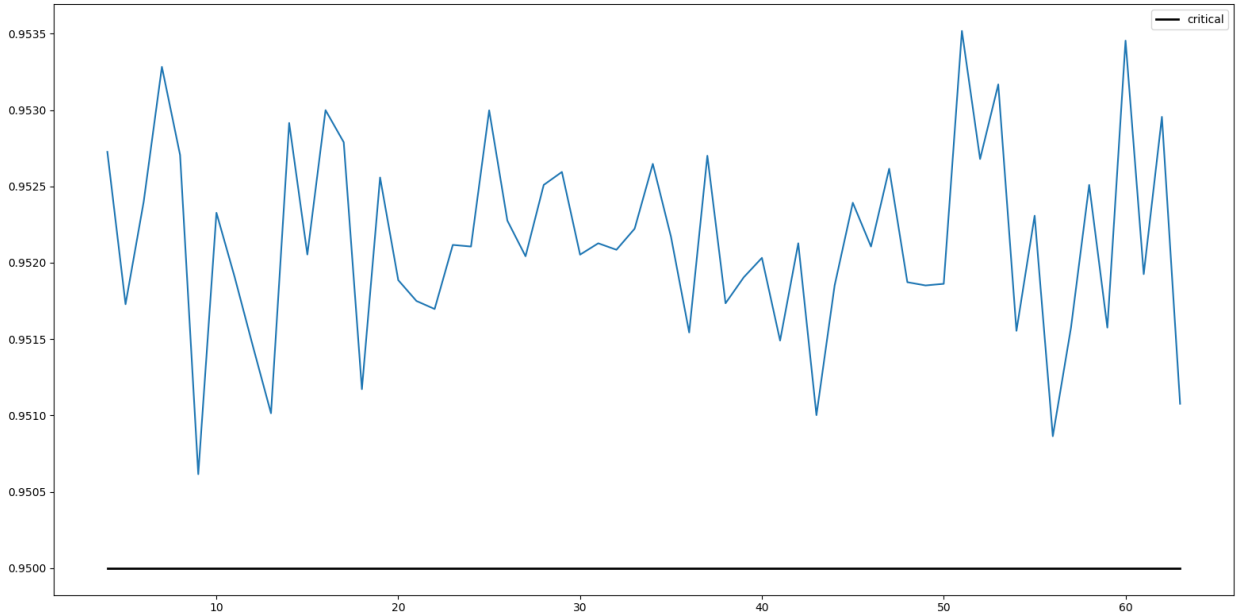


Рисунок 6.1 - График зависимости количества пройденных статистических тестов при дешифровании одним ключом от количества ключевых файлов

На графике видно, что при попытке вскрытия шифра одним файлом из набора сессионных, характеристики случайности зашифрованной последовательности не ухудшаются, а процент прохождения последовательностью статистических тестов находится до отметки 5%, являющейся достаточной для доказательства надежности шифра.

Также, результаты прохождения статистического теста при вскрытии шифртекста одним ключом, были разделены по принципу использующегося при силовой атаке файла: сессионный файл или сессионный ключ шифрования (рис. 6.2).

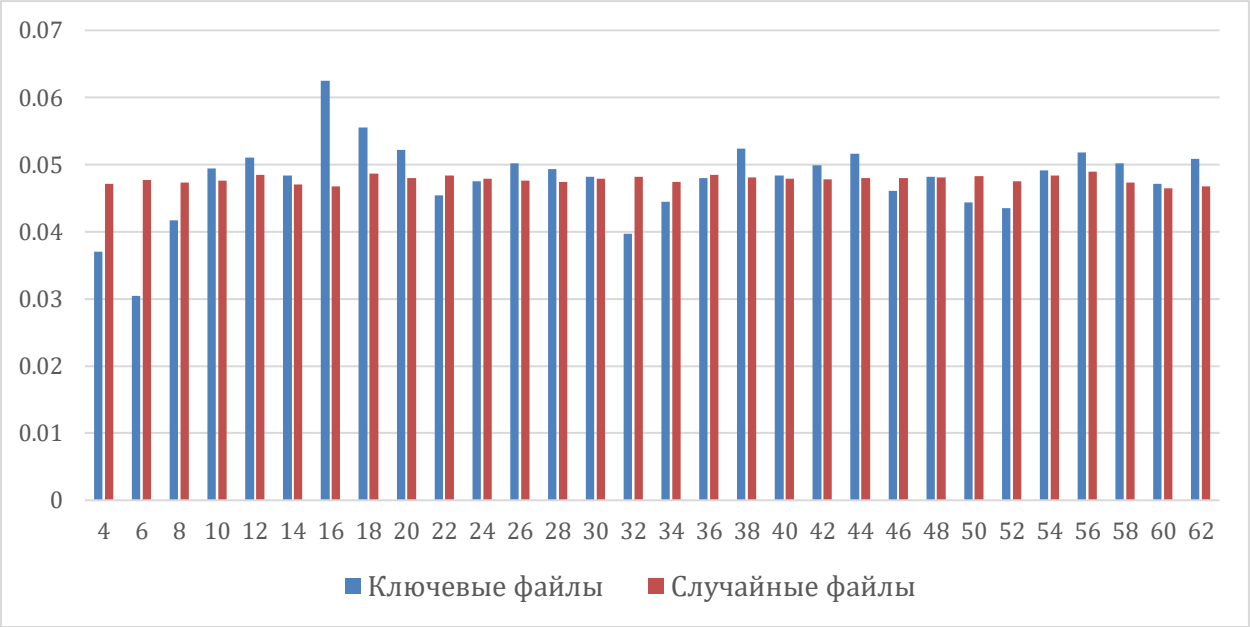


Рисунок 6.2 - График зависимости количества пройденных статистических тестов от количества ключевых файлов

Из данных графика видно, что процент вероятности появления случайной последовательности при использовании для вскрытия ключевого файла приблизительно равен проценту вероятности при использовании не ключевого файла. Данное свойство алгоритма шифрования усложняет криптоаналитику задачу вскрытия, потому что нельзя предсказать, был ли файл, вносящий статистические зависимости в последовательность, ключевым.

6.2 Применение силовой атаки при помощи некоторого процента ключевых файлов

Для того, чтобы провести исследование результатов применения силовой атаки при помощи нескольких файлов, включая некоторый процент ключевых, были проведены следующие действия:

1. Количество сессионных ключей, использующихся при непосредственном шифровании файла равно n, где $n = 4 \ldots 40$.
2. Файл с открытым текстом шифруется при помощи n ключей шифрования.
3. Выбираются n сессионных файлов, при помощи которых будет произведена попытка взлома. Файлы для вскрытия шифра выбираются следующим

образом: выборка из n файлов содержит i ключевых файлов и n-i сессионных файлов, при $i = 0…n$.

4. К зашифрованной последовательности и выбранным файлам для вскрытия применяется операция логического сложения по модулю два.
5. Полученная в результате попытки вскрытия шифра последовательность проходит статистический тест, чтобы определить были ли снижены статистические характеристики последовательности.

Данное исследование рассчитано на то, чтобы выявить увеличится ли вероятность вскрытия шифртекста, если криптоаналитик будет владеть информацией о некоторых использующихся при шифровании ключевых файлах. Определить является ли достоверным утверждение, что чем больше ключевых файлов определил криптоаналитик, тем больше вероятность найти оставшиеся ключевые файлы.

Было произведено 1000 сессий шифрования при помощи различных случайный сессионных файлов, полученные значения были усреднены и отображены в графике зависимости количества пройденных статистических тестов от процента ключевых файлов, использующихся при попытке вскрытия зашифрованной последовательности (рис. 6.3).

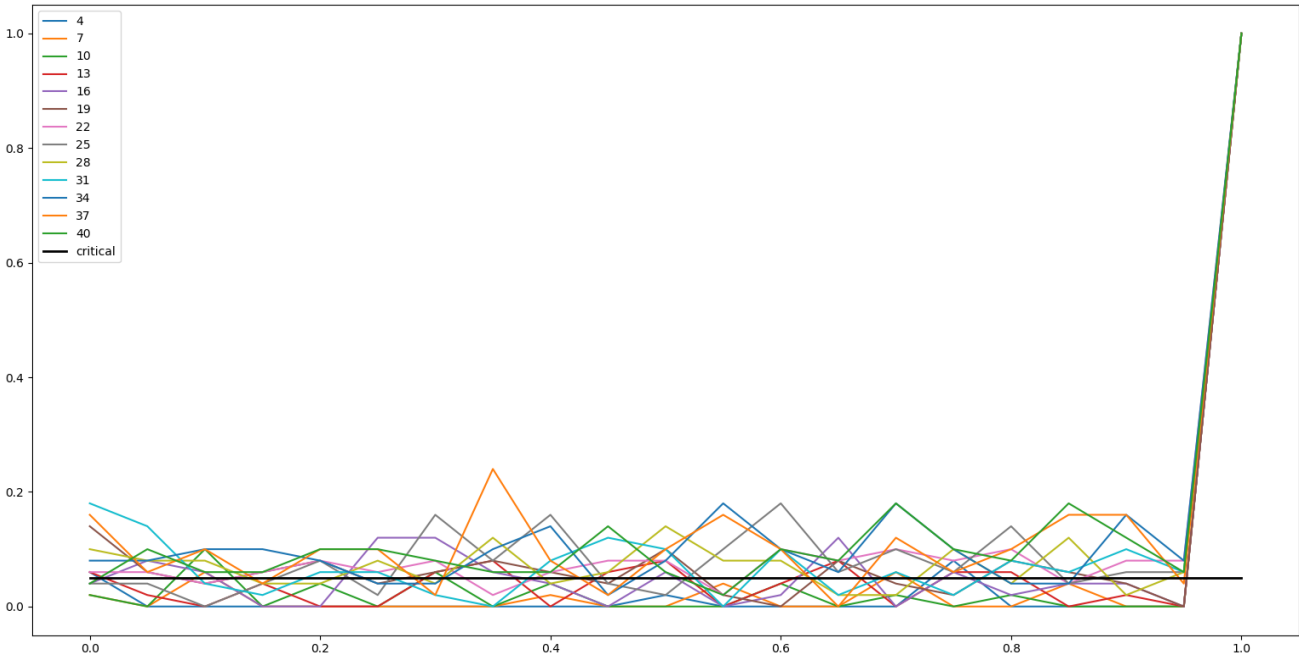


Рисунок 6.3 - График зависимости количества пройденных статистических тестов от процента ключевых файлов, использующихся при попытке вскрытия зашифрованной последовательности

Исходя из полученного графика можно сделать вывод, что вероятность получения неслучайной последовательности не зависит от того, какой процент ключевых файлов от файлов, использующихся при вскрытии, был использован (за исключением ситуации со 100% ключевыми файлами).

Данное свойство алгоритма шифрования усложняет криптоаналитику задачу вскрытия, потому что увеличение процента ключевых файлов в выборке файлов для вскрытия не ведет к увеличению вероятности появления статистических зависимостей. Соответственно, криптоаналитик не сможет отличить ситуацию, когда им было выбрано 90% ключевых файлов от ситуации, когда было выбрано 100% случайных файлов.

7. Аналитические атаки

7.1 Применение метода корреляционной атаки

Корреляционные атаки проводятся для того, чтобы выявить информацию о некоторых внутренних компонентах генератора. Впоследствии, криптоаналитик производит исследование полученной информации о генераторе гаммирующих последовательностей, опираясь на эти знания атаку на шифр можно будет реализовать за время много меньшее полного перебора. Предполагается, что криптоаналитику известно описание генератора, он обладает открытым и соответствующим ему закрытым текстом. Задача криптоаналитика – определение применяемого ключа или начального заполнения регистров.

К шифру, основанному на алгоритме Шеннона, невозможно применить атаки подобного типа. Криптоаналитик, получив открытый и закрытый текст может при помощи операции обратного преобразования, сложение по модулю два между двумя имеющимися последовательностями, определить часть гаммы, при помощи которой был зашифрован конкретный текст. Но, чтобы имея гамму криптоаналитик смог узнать при помощи какого количества и непосредственно каких файлов из общего количества сессионных файлов, была получена шифрующая последовательность (начальное заполнение), ему необходимо перебрать всевозможные комбинации из 1024 сессионных файлов, что является по сути полным перебором ключей.

7.2 Применение атаки методом компромисс время-память

Целью данного алгоритма также является выявление начального заполнения состояния регистра шифрующей гамма-последовательности. Атака состоит из этапа формирования словаря, содержащего все возможные пары “состояние-выход”, и этапа создания некоторого предполагаемого начального заполнения. После чего из этих данных генерируется выход для нахождения совпадения с перехваченной последовательностью.

Данную атаку нецелесообразно применять к исследуемому алгоритму шифрования, так как пространство внутренних состояний шифра слишком велико для перебора и проверки всех возможных внутренних состояний.

8. ВЫВОДЫ

В результате проведенных исследований, направленных на изучение надежности шифра на основе алгоритма Шеннона, можно сделать следующие выводы:

1. При попытке обнаружить статистические зависимости в результате сложения по модулю два зашифрованной последовательности и одного из сессионных файлов, нельзя предсказать, был ли файл, вносящий статистические зависимости в последовательность, ключевым.
2. Невозможно выявить ключевые файлы, использовавшиеся при создании шифртекста, через обнаружение статистических связей, так как вероятность получения неслучайной последовательности не зависит от того, какой процент ключевых файлов от файлов, использующихся при вскрытии, был использован.
3. Криптоаналитические атаки по выявлению начального заполнения регистров не принесут результата, так как при имеющихся данных, будь то часть шифрующей гаммы, либо открытый и закрытый тексты, задача определения начального заполнения сводится к полному перебору всех возможных сессионных файлов.

На основе проведенных тестов, шифр на основе алгоритма Шеннона можно считать криптографически стойким в случае, когда были соблюдены условия шифрования открытого текста: выбранные ключевые файлы являются случайными последовательностями, длины ключевых файлов являются взаимно простыми числами.

ЗАКЛЮЧЕНИЕ

При выполнении дипломной работы, было проведено исследование надежности шифра, построенного на одном из алгоритмов Шеннона.

В ходе исследования были разобраны следующие методы криптоанализа поточных шифров: статистический анализ, силовые атаки, аналитические атаки, корреляционная атака, атака методом компромисс время-память. Был изучен алгоритм выявления статистических зависимостей внутри последовательности стопка книг. Полученные знания были применены при анализе надежности шифра.

Был написан набор вспомогательных модулей, направленных на реализацию следующих целей:

- шифрование и дешифрование последовательности при выбранном количестве ключевых файлов
- выявление статистических свойств полученных последовательностей
- осуществление алгоритма силовой атаки к шифртексту, применяя один текст из имеющиеся сессионных файлов
- осуществление алгоритма силовой атаки к шифртексту, применяя набор из нескольких сессионных файлов, варьируя процент ключевых файлов
- обработка и анализ результатов проведенных тестов, представление полученных итогов в графическом виде.

Полученные в ходе проведения результатов эксперименты были проанализированы, на основе полученных значений были построены графики зависимости отображающие основные итоги тестирования.

Теоретически исследованы возможности проведения как атак методом грубой силы, так и криптоаналитических атак. Были изучены и теоретически применены корреляционная атака и атака компромисс время-память.

В результате анализа шифра Шеннона с применением различных методов криптоанализа, можно сказать о том, что ни одна из проведенных на шифр атак не является менее трудоемкой, чем атака полного перебора сессионных файлов для получения шифрующей гамма последовательности.

Основываясь на результатах проведенного исследования надежности шифра, были более подробно изучены некоторые его свойства, влияющие на криптостойкость алгоритма шифрования. Также теоретически и практически доказана защищенность рассматриваемого алгоритма шифрования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Романец Ю.В. Защита информации в компьютерных системах и сетях / Ю. В. Романец, П. А. Тимофеев, В. Ф. Шаньгин. - 2-е издание – М.: Радио и связь, 2001. – 376 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. Панасенко С. П. Алгоритмы шифрования. Специальный справочник / С. П. Панасенко. – СПб.:БХВ-Петербург, 2009. – 576 с.
4. Архангельская А.В., Запечников С.В. Характеристики области эффективного применения методов поточного шифрования для защиты трафика в телекоммуникационных системах // Информационное противодействие угрозам терроризма. 2005. № 4. С. 183-186.
4. Петров А. А. Компьютерная безопасность. Криптографические методы защиты / А. А. Петров. – М.: ДМК Пресс, 2008. – 448 с.
5. А. И. Пестунов Статистический анализ современных блочных шифров // Вычислительные технологии. – 2007. – т.12, №2 – С. 122-130.
6. Ковтун В.Ю. Криптоанализ симметричных криптосистем: поточные шифры. Криптоанализ по побочным каналам NRJETIX, 2000 - 2008.- 6 с.
7. Стасев Ю.В., Потий А.В, Избенко Ю.А. Исследование методов криптоанализа поточных шифров. - 19 с.
8. Абзалов В.Ш. Анализ методов криптоанализа поточных шифров // Молодежный научный форум: Технические и математические науки: электр. сб. ст. по мат. XXXVIII междунар. студ. науч.-практ. конф. № 9(38). URL: https://nauchforum.ru/archive/MNF_tech/9(38).pdf (дата обращения: 13.03.2019)
9. 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 (дата обращения: 21.10.2018)

10. Biryukov, A., Shamir, A., Wagner, D.: Real Time Cryptanalysis of A5/1 on a PC. In: Schneier, B. (ed.) FSE 2000. LNCS, vol. 1978, pp. 1–18. Springer, Heidelberg (2001)

ПРИЛОЖЕНИЕ 1 – Реализация основного класса для проведения силовых атак

```
import sys
from random import choice
from string import ascii_lowercase
from src.protocol.src.shannon import Shannon
from test_.book.check import BookStack
from itertools import combinations


class Bruteforce:
    def __init__(self):
        self.count_session = 50  # Количество сессий тестирования
        self.session = None  # Номер сессии
        self.count_files = 1  # количество файлов для тестирования
        self.files = None  # Выбранные файлы сессии
        self.keys = None  # Файлы ключей сессии используемые для шифрования
        self.test_file = 'brute_file'  # Файл для результата тестирования
        self.out_file_name = ''  # Название выходного файла
        self.N = 1  # Количество тестов стопка книг

    @staticmethod
    def xor(answer, text):
        if text:
            for i in range(len(answer)):
                answer[i] = answer[i] ^ text[i % len(text)]
        return answer

    def double_files(self, files):
        data, double_keys = [], []
        for first in range(1, len(files)+1):
            for second in range(first + 1, len(files)+1):
                with open(r"../../helper/text/%s" % first, 'rb') as f:
                    data.append(bytearray(f.read()))
                with open(r"../../helper/text/%s" % second, 'rb') as f:
                    data.append(bytearray(f.read()))
                double_keys.append(self.xor(data[0], data[1]))
        return double_keys
```

```
    @staticmethod
    def single_files(files):
        single_key = []
        for file in files:
            with open(r"../../helper/text/%s" % file, 'rb') as f:
                single_key.append(bytearray(f.read()))
        return single_key

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

    def _make_key(self, exploit):
        """
        Метод из множества ключей формирует выходной ключ
        :param exploit:
        :return:
        """
        new_key = bytearray()
        for key in exploit:
            with  open('%s\\..\\..\\helper\\text\\%s'  %  (sys.path[0],  key),  'rb')  as
file_exploit:
                text_exploit = bytearray(file_exploit.read())
                new_key = self._code_text(text_exploit, new_key)
        return new_key

    def _make_exploit(self, exploit):
        """
        Метод формирует из множества ключей шифр последовательность
        :param exploit:
        :return:
        """
        if isinstance(exploit, str):
            exploit = [exploit]
        return self._make_key(exploit)

    def _code_file(self, file_in, file_out, exploit):
        text_exploit = self._make_exploit(exploit)
        with open('%s\\%s' % (sys.path[0], self.test_file), 'wb') as file_result:
```

```
            with open('%s' % file_out, 'rb') as file_code:
                while True:
                    text = file_code.read(1024)
                    if text == b'':
                        break
                    text = self._code_text(text, text_exploit)
                    file_result.write(text)

    def _switch(self, text, file_in=None, file_out=None, exploit=None):
        text_out = None
        if file_in and file_out and exploit:
            self._code_file(file_in, file_out, exploit)
        elif text and exploit and isinstance(exploit, str):
            text_out = self._code_text(text, exploit)
        return text_out

    def get_result(self, mode, check_status, exploit, info=None, file_in=None):
        f = file_in or open(r"log_test/%s/session_%s_%s" % (mode, self.out_file_name,
self.session), 'a')
        if info:
            f.write('Тест: {}\n'.format(info))
        if not check_status and mode == 'single':
            f.write('%s %s %s\n' % (
                str(check_status),
                str(exploit),
                str('..\\..\\helper\\text\\'+exploit in self.keys)
            ))
        elif not check_status and mode != 'single':
            f.write('%s %s\n' % (
                str(check_status),
                str(exploit)
            ))
        else:
            f.write('%s\n' % str(check_status))

    def _brute_all(self, text, file_in, file_out, count_files_key=4, session=1):
        """
        Методя для перебора всех файлов, входящих в список файлов сессии
        :param text:
        :param file_in:
```

```
        :param file_out:
        :return:
        """
        self.session = session
        # print('-' * 30)
        # print('Запуск сессии "%d"' % session)
        self.out_file_name = file_out.split('\\')[-1]
        self.test_file = 'temp\\test_%s' % self.out_file_name
        cipher, self.files, self.keys = Shannon(count_files_key).encode(text, file_in,
file_out)
        with open(r"log_test/%s/session_%s_%s_keys_%d" % ('single', self.out_file_name,
self.session, count_files_key), 'a') as f:
            f.write('\n\nЗапуск сессии {}\n'.format(session))
            for exploit in self.files:
                self._switch(text, file_in, file_out, exploit)
                self._check_result(exploit, 'single', f)

    def _brute_key_file(self, text, file_in, file_out, count_files_key=4, session=1):
        """
        Метод для перебора файлов, входящих в список ключей
        :param text:
        :param file_in:
        :param file_out:
        :return:
        """
        self.session = session
        self.out_file_name = file_out.split('\\')[-1]
        self.test_file = 'temp\\test_%s' % self.out_file_name
        cipher, self.files, self.keys = Shannon(count_files_key).encode(text, file_in,
file_out)
        with open(r"log_test/%s/session_%s_%s_keys_%d" % ('key', self.out_file_name,
self.session, count_files_key), 'a') as f:
            f.write('\n\nЗапуск сессии {}\n'.format(session))
            for exploit in self.keys:
                self._switch(text, file_in, file_out, exploit)
                self._check_result(exploit, 'key', f)

    def _brute_keys_files(self, text, file_in, file_out, count_files_key=4,
count_exploit_file=1, session=1):
        """
```

```
        Метод для перебора нескольких файлов, входящих в список ключей
        :param text:
        :param file_in:
        :param file_out:
        :return:
        """
        self.session = session
        # print('-' * 30)
        # print('Запуск сессии "%d"' % session)
        self.out_file_name = file_out.split('\\')[-1]
        self.test_file = 'temp\\test_%s' % self.out_file_name
        cipher, self.files, self.keys = Shannon(count_files_key).encode(text, file_in,
file_out)
        with open(r"log_test/%s/session_%s_%s_key_%d" %
                  ('keys', self.out_file_name, self.session, count_exploit_file), 'a') as
f:
            f.write('\n\nЗапуск сессии {}\n'.format(session))
            for exploit in combinations(self.keys, count_exploit_file):
                self._switch(text, file_in, file_out, exploit)
                self._check_result(exploit, 'keys', f)

    def _check_result(self, exploit, path='single', file_in=None):
        """
        Метод запускает N раз тестирование последовательности, для получения точного
результата
        :param exploit:
        :return:
        """
        for i in range(self.N):
            check_result = BookStack().check(file=self.test_file)
            info = None
            if isinstance(exploit, str):
                if '..\\..\\helper\\text\\{}'.format(exploit) in self.keys:
                    info = 'Файл {} входит в последовательность, которой
шифровали'.format(exploit)
                else:
                    info = 'Файл {} не входит в последовательность, которой
шифровали'.format(exploit)
            self.get_result(path, check_result[1], exploit, info, file_in)
```

```
    def test(self, text, file_in=None, file_out=None):
        """
        Метод для запуска теста брутфорса
        :param text:
        :param file_in:
        :param file_out:
        :return:
        """
        if text:
            text = bytearray(text.encode())
        for count_files_key in range(16, 256):
            for session in range(self.count_session):
                self.session = session
                print('Запуск сессии "%d" с ключами %d' % (session, count_files_key))
                self._brute_all(text, file_in, file_out, count_files_key, session)


if __name__ == '__main__':
    _text = None
    _file_in = '%s\\test3' % sys.path[0]
    _out_name = ''.join([choice(ascii_lowercase) for i in range(10)])
    _file_out = '%s\\temp\\%s' % (sys.path[0], _out_name)
    Bruteforce().test(_text, _file_in, _file_out)
```

ПРИЛОЖЕНИЕ 2 – Анализ статистических свойств шифртекстов

```
from os import listdir
from matplotlib import pylab

files = listdir('log_test/not_exp')
n, m = 1, 100
statistic_not_exp = {i: 0 for i in range(n, m)}
for file in files:
    with open('log_test/not_exp/{}'.format(file), 'r') as f:
        false_file, true_file = 0, 0
        line = f.readline()
        while line:
            if 'True' in line:
                true_file += 1
            elif 'False' in line:
                false_file += 1
            line = f.readline()
        encoding_files = int(file.split('_')[-1])
        statistic_not_exp[encoding_files] = true_file / (true_file + false_file)

for key in sorted(statistic_not_exp.keys()):
    print(key, ': ', statistic_not_exp[key])

x = [i for i in range(n, m)]
y = [0.95 for i in range(len(x))]

pylab.plot(list(statistic_not_exp.keys()), list(statistic_not_exp.values()))
pylab.plot(x, y, label='critical', linewidth=2.0, color='black')
pylab.legend()
pylab.show()
```

ПРИЛОЖЕНИЕ 3 – Анализ статистических свойств при наложении на шифртекст одного сессионного файла

```
from os import listdir
from re import findall
from matplotlib import pylab

files = listdir('log_test/single')
result = {}
count_files = {}
answer = {}
n, m = 4, 64
index = 1
for file in files:
    with open('log_test/single/{}'.format(file), 'r') as f:
        data = f.read()
        test = findall(r'(True|False [\d]+ [\w]*)', data)
        result[index] = test
        count_files[index] = int(file.split('_')[-1])
        answer[count_files[index]] = []
        index += 1
trues = {}
lens = {}
keys_file = {}
not_keys_file = {}
statics = {}
for key in result.keys():
     # Сколько успешно пройденных тестов
    trues[key] = len([_ for _ in result[key] if _ == 'True'])      lens[key] = len([_ for
_ in result[key]])  # Сколько всего тестов было в комплекте
    # Сколько тестов было взломано файлами не из списка ключей
    not_keys_file[key] = len([_ for _ in result[key] if _[-5:] == 'False'])
    # Сколько тестов было взломано файлами из списка ключей
    keys_file[key] = len([_ for _ in result[key] if _[-4:] == 'True' and len(_) > 4])
    answer[count_files[key]].append((
        trues[key],
```

```
        lens[key],
        trues[key] / (lens[key] ),
        keys_file[key] / count_files[key],
        not_keys_file[key] / (1024 - count_files[key])
    ))
average_percentage = {i: 0 for i in range(n, m)}
average_key = {i: 0 for i in range(n, m)}
average_random = {i: 0 for i in range(n, m)}
for key in range(n, m):
    for session in answer[key]:
        average_percentage[key] += (session[2] / len(answer[key]))
        average_key[key] += (session[3] / len(answer[key]))
        average_random[key] += (session[4] / len(answer[key]))

x = [i for i in range(n, m)]
y = [0.95 for i in range(len(x))]
pylab.plot(list(average_percentage.keys()), list(average_percentage.values()))
pylab.plot(x, y, label='critical', linewidth=2.0, color='black')

pylab.legend()
pylab.show()
```

ПРИЛОЖЕНИЕ 4 – Анализ статистических свойств при вскрытии шифртекста некоторым количеством ключей

```
from os import listdir
from matplotlib import pylab
from math import floor

files = listdir('log_test/percentage_decrypted_keys/key')
result = {}
count_files = {}
answer = {}

n, m = 4, 41
statistics_decrypted = {}

for file in files:
    with open('log_test/percentage_decrypted_keys/key/{}'.format(file), 'r') as f:
        key_encrypted_counts = int(file.split('_')[-5])
        key_decrypted_counts = int(file.split('_')[-3])
        random_decrypted_counts = int(file.split('_')[-1])
        percentage_key_files = round(key_decrypted_counts/key_encrypted_counts, 3)
        percentage_key_files = floor(percentage_key_files*20)/20
        data = f.read()
        if data.startswith('False'):
            statistics_break = 1
        else:
            statistics_break = 0

        if statistics_decrypted.get(key_encrypted_counts, None) is None:
            statistics_decrypted[key_encrypted_counts] = {i/100: 0 for i in range(0, 105,
5)}

        if  statistics_decrypted[key_encrypted_counts].get(percentage_key_files,  None)  is
None:
```

```
            statistics_decrypted[key_encrypted_counts][percentage_key_files]             =
statistics_break / 50
        else:
            statistics_decrypted[key_encrypted_counts][percentage_key_files]            +=
statistics_break / 50

x = [i/100 for i in range(0, 105, 5)]
y = [0.05 for i in range(len(x))]
# for key, value in statistics_decrypted.items():
for key in range(n, m, 3):
    pylab.plot(list(statistics_decrypted[key].keys()),
list(statistics_decrypted[key].values()), label=str(key))
pylab.plot(x, y, label='critical', linewidth=2.0, color='black')
pylab.legend()
pylab.show()
```