Федеральное бюджетное государственное образовательное учреждение

«Новосибирский государственный университет

экономики и управления «НИНХ»

На правах рукописи

Перов Артём Андреевич

УНИВЕРСАЛЬНЫЙ МЕТОД ПОСТРОЕНИЯ РЕШАЮЩИХ ПРАВИЛ С ИСПОЛЬЗОВАНИЕМ СВЕРТОЧНЫХ НЕЙРОННЫХ СЕТЕЙ ДЛЯ АНАЛИЗА ГЕНЕРАТОРОВ ПСЕВДОСЛУЧАЙНЫХ ПОСЛЕДОВАТЕЛЬНОСТЕЙ НА ОСНОВЕ ИТЕРАТИВНЫХ БЛОЧНЫХ ШИФРОВ

05.13.17 – Теоретические основы информатики

Диссертация на соискание учёной степени кандидата технических наук

Научный руководитель:

кандидат физико-математических наук,

доцент Пестунов Андрей Игоревич

Новосибирск – 2020

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Актуальность и степень разработанности проблемы. Псевдослучайные последовательности имеют большое значение при решении многих задач, связанных с исследованием информационных моделей, анализом функционирования программно-аппаратных средств, обеспечением высоконадежной обработки информации, в том числе для целей ее передачи, хранения и защиты.

Особым классом генераторов псевдослучайных последовательностей являются генераторы с итеративной структурой, формирующие очередное псевдослучайное число посредством повторения относительно простого преобразования, называемого раундом, над входными данными несколько раз. Зачастую такие генераторы разрабатываются на основе итеративных блочных шифров в режиме счетчика (CTR).

Оценка качества генераторов обычно осуществляется через применение методов обнаружения закономерностей и отклонений от случайности полученных псевдослучайных последовательностях. Выбор подходов к построению решающих правил для обнаружения таких закономерностей обусловлен итеративной структурой рассматриваемых генераторов. Для этого разрабатываются специальные решающие правила на основе так называемых различителей, предназначенных для распознавания псевдослучайных последовательностей, получаемых после разного количества раундов. При этом важной задачей является поиск максимального количества раундов, при котором такие правила способны отличить разные раунды друг от друга. Эффективные решающие правила на основе различителей представляют научный интерес как сами по себе, так и в комплексе, когда на их основе создаются алгоритмы вычисления неизвестных параметров генератора. В случае применения блочного шифра это могут быть секретные ключи.

Методы построения решающих правил на основе различителей можно условно разделить на два класса: аналитические и эмпирические (в основном статистические). Многие аналитические методы базируются на выявлении дифференциальных, линейных или интегральных признаков, описывающих свойства псевдослучайной последовательности после заданного числа раундов. Затем обнаружение закономерностей осуществляется через выявление этих свойств в сгенерированной последовательности.

Большой вклад в развитие аналитических методов построения решающих правил на основе различителей внесли зарубежные ученые A. Shamir, E. Biham, A. Biruykov, B. Schneier, M. Matsui, D. Wagner и др. Ими предложены разнообразные подходы к построению различителей, а также алгоритмы, позволяющие вычислять на их основе ключи шифрования. Аналитические подходы характерны тем, что позволяют строить атаки на достаточно большое количество раундов, давая возможность находить уязвимости, которые проявляются только на огромных выборках или при использовании алгоритмов, требующих огромных вычислительных ресурсов, недоступных на практике (например, порядка 2^{128} элементарных операций или байт оперативной памяти). Однако поскольку используемые в различителях признаки тесно связаны с конкретными шифрами, то аналитически построенные решающие правила не являются универсальными и эффективны только для целевого шифра.

Помимо аналитических методов построения решающих правил для обнаружения закономерностей в псевдослучайных последовательностях, полученных при помощи генераторов на основе итеративных блочных шифров, могут использоваться эмпирические статистические методы. В рамках таких методов решающие правила строятся на основе критериев, позволяющих отличить последовательность от случайной в ходе экспериментов на выборках, размер которых приемлем для расчетов. Так, в работах L. Knudsen предложена и для некоторых генераторов псевдослучайных последовательностей успешно применена универсальная

методика вычисления неизвестных параметров генератора, где в качестве различителя выступает статистический критерий хи-квадрат. Для малого числа раундов, когда для распознавания отклонения от равномерного распределения достаточно выборок, размеры которых относительно невелики, атака осуществляется экспериментально, а для большего числа раундов размер необходимой выборки экстраполируется аналитически на основе полученных экспериментальных данных.

Достоинством решающих правил на основе статистических различителей является их универсальность, проявляющаяся в том, что по одной и той же схеме можно проанализировать серию генераторов псевдослучайных последовательностей без детального учета индивидуальных особенностей архитектуры каждого из них. При этом необходимость проведения экспериментальных расчетов накладывает ограничения на размер выборки, которую возможно обработать. Применение машинного обучения имеет потенциал добиться снижения размера выборки за счет более тонкого анализа паттернов, встречающихся в псевдослучайных последовательностях, полученных при варьируемом количестве раундов итеративного блочного шифра, на котором основан генератор.

Кроме того, решающие правила на основе статистических различителей далеко не всегда способны обнаруживать сложные паттерны в проверяемых выборках и использовать это при выявлении отклонений от случайности. Обычно решение принимается на базе интегральных накопительных характеристик, обновляемых после обработки очередного выборочного элемента, но не учитывающих многие корреляции.

Машинное обучение позволяет решать широкий спектр задач по реализации систем поддержки принятия решений, прогнозированию, оптимизации и распознаванию образов. Эти технологии уже применялись к исследованию генераторов псевдослучайных последовательностей на основе итеративных блочных шифров, однако в основном такие методы используют не только сгенерированные псевдослучайные числа, но и требуют

дополнительных данных, полученных через побочные каналы (см. работы L. Lerman, G. Bontempi, B. Hettwer, S. Gehrer и др.).

M. Bernardi, P. Malacaria и др. показали, что глубокая нейронная сеть способна обучиться генерировать псевдослучайные последовательности так, чтобы они отвечали требованиям информационной безопасности и проходить ряд тестов на случайность. Наиболее эффективные решающие правила для обнаружения отклонений от случайности, вызванных скрытой в стего-изображениях информации, также активно используют именно машинное обучение, в частности, метод опорных векторов и ансамблевые классификаторы (см. работы J. Fridrich, A. Ker, M. Goljan, T. Pevny, J. Kodovsky, R. Bohme и др.). Такая информация может быть обнаружена за счет того, что она, хотя и незначительно, но нарушает статистические связи между соседними пикселями стего-контейнера.

Таким образом, применение методов машинного обучения имеет потенциал их использования в разработке решающих правил для обнаружения закономерностей и отклонений от случайности в псевдослучайных последовательностях, которые, с одной стороны, обладают универсальностью, а с другой – учитывают внутреннюю структуру итеративных генераторов.

Рабочая гипотеза исследования. Решающие правила на основе сверточных нейронных сетей могут позволить обнаруживать закономерности и отклонения от случайности в псевдослучайных последовательностях, полученных посредством генераторов на основе итеративных блочных шифров более эффективно, чем универсальные решающие правила на базе статистических тестов.

Целью диссертационной работы является разработка универсального метода построения решающих правил на основе сверточных нейронных сетей для обнаружения закономерностей и отклонений от случайности в псевдослучайных последовательностях, полученных посредством генераторов, созданных на основе итеративных блочных шифров.

Для достижения цели решались следующие **задачи:**

1. Разработать алгоритм обработки псевдослучайных последовательностей для представления их в формате, пригодном для обучения нейронной сети.

2. Разработать, алгоритмически описать и математически обосновать метод построения решающих правил для обнаружения закономерностей и отклонений от случайности в псевдослучайных последовательностях, полученных при помощи генераторов на базе блочных шифров.

3. Создать программные реализации разработанных алгоритмов и применить их к экспериментальному исследованию генераторов псевдослучайных последовательностей на базе современных итеративных блочных шифров; сравнить полученные результаты с результатами анализа псевдослучайных последовательностей посредством универсальных решающих правил на основе статистических тестов.

4. Определить параметры генераторов на базе итеративных блочных шифров, обеспечивающие неотличимость полученных псевдослучайных последовательностей от равномерно распределенных случайных величин.

Объектом исследования являются генераторы псевдослучайных последовательностей на основе итеративных блочных шифров.

Предметом исследования являются решающие правила для обнаружения закономерностей в псевдослучайных последовательностях, полученных посредством генераторов с итеративной структурой.

Методы исследований. Методы машинного обучения, сверточные нейронные сети; аппарат теории вероятностей и математической статистики; технологии структурного и объектно-ориентированного программирования; инструментарий графического моделирования и визуализации данных.

Научная новизна диссертационной работы заключается в следующем.

1. Предложен и теоретически обоснован новый метод построения решающих правил на основе свёрточных нейронных сетей для обнаружения закономерностей в псевдослучайных последовательностях, полученных с

помощью итеративных генераторов. В отличие от аналитических подходов, которые предполагают анализ внутренней структуры конкретного генератора и поэтому не применимы к другим, новый метод универсален и позволяет строить решающие правила для любых итеративных генераторов. В то же время, в отличие от многих универсальных правил, основанных на статистических критериях и обрабатывающих выборочные значения отдельно друг от друга, нейронная сеть принимает для анализа всю выборку целиком, что дает возможность более точного обнаружения закономерностей.

2. Экспериментально подтверждена эффективность построенных решающих правил применительно к генераторам псевдослучайных чисел, основанных на итеративных блочных шифрах в режиме счетчика. Для ряда генераторов построенные решающие правила позволяют достичь лучших результатов по сравнению со статистическими тестами, в том числе, базирующимися на адаптивных кодах и структурах («стопка книг», порядковый тест и адаптивный критерий хи-квадрат) и др. В частности, решающие правила позволяют обнаруживать закономерности (отклонения от равномерного распределения) при меньших объемах выборок и при большем количестве итераций генератора.

3. В результате применения построенных решающих правил для ряда генераторов на основе современных итеративных блочных шифров уточнены существующие оценки минимального количества итераций (раундов), которое требуется для обеспечения удовлетворительных статистических свойств псевдослучайных последовательностей, а также впервые получены новые оценки для тех генераторов, для которых таких оценок не существовало.

Положения, выносимые на защиту.

1. Метод построения универсальных решающих правил для обнаружения закономерностей в псевдослучайных последовательностях.

2. Теоретическое обоснование предложенного метода и результаты экспериментального исследования, подтверждающие его эффективность.

3. Оценки минимального количества раундов, требуемого для обеспечения удовлетворительных статистических свойств псевдослучайных последовательностей, полученных с помощью итеративных генераторов на основе блочных шифров.

4. Созданный на основе предложенного метода программный комплекс для анализа псевдослучайных последовательностей, полученных при помощи итеративных генераторов, основанных на блочных шифрах.

Теоретическая значимость. Разработан новый универсальный метод построения решающих правил на основе сверточных нейронных сетей для анализа генераторов псевдослучайных последовательностей на основе итеративных блочных шифров. Метод имеет перспективу быть развитым в общий подход построения решающих правил для анализа любых генераторов на основе итеративных алгоритмов, в том числе хеш-функций и других.

Практическая значимость работы. Разработан программный комплекс для статистического анализа генераторов псевдослучайных последовательностей на базе итеративных блочных шифров при помощи решающих правил на основе сверточных нейронных сетей и универсальных статистических тестов. Для ряда генераторов определены параметры, обеспечивающие удовлетворительные статистические свойства получаемых псевдослучайных последовательностей и отсутствие закономерностей в них. Получены два свидетельства о регистрации программ для ЭВМ в Федеральной службе по интеллектуальной собственности.

Результаты диссертационной работы используются в образовательном процессе кафедры информационных технологий ФГБОУ ВО НГУЭУ и факультета информационных технологий ФГАОУ ВО НГУ, а также в практической деятельности компании «Акстел-Безопасность» и Научно-исследовательского института информационно-коммуникационных технологий (НИИ ИКТ).

Достоверность результатов обеспечена корректностью постановок задач, математическими доказательствами теоретических утверждений, экспериментальной проверкой теоретических результатов, сравнением полученных экспериментальных данных с эталонными. Соответствие диссертации паспорту научной специальности.

Содержание диссертации соответствует п. 5 («Разработка и исследование моделей и алгоритмов анализа данных, обнаружения закономерностей в данных и их извлечениях; разработка и исследование методов и алгоритмов анализа текста, устной речи и изображений»), п.7 («Разработка методов распознавания образов, фильтрации, распознавания и синтеза изображений, решающих правил. Моделирование формирования эмпирического знания»), п. 11 («Разработка методов обеспечения высоконадежной обработки информации и обеспечения помехоустойчивости информационных коммуникаций для целей передачи, хранения и защиты информации; разработка основ теории надежности и безопасности использования информационных технологий»).

Апробация работы. Результаты диссертации докладывались и обсуждались на следующих конференциях и семинарах: XIV Международная научно-практическая конференция «Информационная безопасность», Таганрог (2015); Всероссийская конференция молодых ученых по математическому моделированию и информационным технологиям (2015, 2019 – получен диплом победителя конференции); Всероссийская научная конференция молодых ученых «Наука. Технологии. Инновации» (2015, 2016); Международная научная студенческая конференция МНСК, (2016, 2017); Научная сессия ИТФ НГУЭУ, секция «Информационная безопасность и защита информации», Новосибирск, (2015, 2016); конф. «Информационные технологии» в рамках науч. сессии НГУЭУ (2017, 2018); Конференция «Актуальные направления научной мысли: проблемы и перспективы», Новосибирск (2018); 18 Всероссийская конференция «Сибирская научная школа-семинар с международным участием "Компьютерная безопасность и

криптография"» SIBECRYPT'19 (2019); 2019 International Multi-Conference on Engineering, Computer and Information Sciences (SIBIRCON), Новосибирск (2019); Международная научно-практическая конференция «Распределенные информационно-вычислительные ресурсы: Цифровые двойники и большие данные» (DICR 2019), Новосибирск (2019);19 Всероссийская конференция «Сибирская научная школа-семинар с международным участием "Компьютерная безопасность и криптография"» SIBECRYPT'20 (2020);

Научный семинар «Криптография и криптоанализ» (рук. к.ф.-м.н. Н.Н. Токарева, ИМ им. С.Л. Соболева СО РАН), 2018, 2019; Научный семинар Сибирского государственного университета телекоммуникаций и информатики (рук. д.т.н. А.Б. Мархасин), 2019; Объединенный семинар ИВТ СО РАН и НГУ «Информационные технологии» (рук. академик Ю.И. Шокин и к.ф.-м.н. А.В. Юрченко), 2020; Научный семинар кафедры «Комплексная защита информации» ОмГТУ (рук. д.т.н. П.С. Ложников), 2020.

Публикации.

По теме диссертации автором опубликовано 18 работ, из них 4 статьи в журналах, которые включены в перечень российских рецензируемых научных журналов и изданий для опубликования основных научных результатов диссертаций, 2 публикации в Scopus/WoS, 11 публикаций в материалах международных и всероссийских конференций, 2 свидетельства о государственной регистрации программы для ЭВМ. Общий объём публикаций составляет 4,36 п.л., авторский вклад – 3,62 п.л.

Объем и структура диссертации.

Диссертационная работа состоит из введения, 3 глав основного содержания, списка использованных источников из 107 наименований и 10 приложений. Общий объем диссертации 153 страницы (основное содержание изложено на 124 страницах), включая 21 иллюстрацию и 12 таблиц.

СОДЕРЖАНИЕ ДИССЕРТАЦИИ

Во **введении** обосновывается актуальность выбранной темы, определены цели и задачи исследования, раскрыта научная новизна работы и практическая значимость полученных результатов. Сформулированы положения, выносимые на защиту.

В **первой главе** представлен обзор исследований по теме диссертации. Рассмотрены современные направления развития машинного обучения, существующие архитектуры нейронных сетей и возможные сферы применения этих технологий для решения практически значимых задач. Проанализированы аналитические и статистические подходы к оцениванию итеративных генераторов псевдослучайных последовательностей, в том числе, на примере блочных шифров.

Во **второй главе** приводится описание метода построения решающих правил на основе свёрточных нейронных сетей для анализа итеративных генераторов псевдослучайных чисел. Предлагаемый метод основан на различении графических эквивалентов псевдослучайных последовательностей, полученных посредством генераторов на основе итеративных блочных шифров.

Третья глава посвящена разработке программной библиотеки и последующему исследованию с её помощью итеративных генераторов из библиотек блочных шифров BLOC и cppcrypto. Предложен подход, позволяющий объединить в единую систему реализации алгоритмов и реализовать методику, основанную на варьировании параметров работы итеративных генераторов псевдослучайных последовательностей. Приведено описание тестов «стопка книг» и «адаптивный критерий хи-квадрат», а также результаты полученные данными тестами. Описаны предлагаемые тесты NIST и приведены результаты, полученные данными тестами для выбранных алгоритмов. Сопоставлены результаты проведенных экспериментов и сделан

вывод о целесообразности использования предложенного во второй главе метода.

В **заключении** сформулированы основные результаты диссертации.

В **приложении** приведены исходные коды библиотек «Униблокс-2015» и её модификации с тестами NIST, а также исходные коды утилит по преобразованию выходных последовательностей генераторов. Представлены акты о внедрении в практическое использование.

Глава 1

Состояние проблем в области итеративных генераторов, криптографических алгоритмов и машинного обучения

В настоящей главе рассматриваются основные характеристики итеративных блочных криптографических алгоритмов, на примере которых исследуются итеративные генераторы псевдослучайных последовательностей, а также общая информация о машинном обучении и его применении в задачах информационной безопасности.

1.1 Статистические исследования над псевдослучайными последовательностями

Потребность в генерации случайных и псевдослучайных чисел возникает во многих задачах, в частности, в криптографических приложениях. Обычно криптосистемы используют ключи, которые должны генерироваться случайным образом. Многие криптографические протоколы также требуют случайных или псевдослучайных входных данных в качестве различных параметров, например, для вспомогательных значений используемых при генерации цифровых подписей или в протоколах аутентификации.

Существует два основных типа генераторов, используемых для создания случайных чисел: генераторы случайных чисел (random number generators или RNGs), и генераторы псевдослучайных чисел (pseudorandom number generators или PRNGs). Для криптографических задач оба этих типа генераторов создают поток нулей и единиц, который может быть разбит на подпотоки и блоки случайных чисел [87].

Случайная битовая последовательность может быть представлена как результат подбрасываний ровной монеты со сторонами «0» и «1», где вероятность выпадения каждой стороны строго равна 0.5. Кроме того, броски являются независимыми: результат любого предыдущего броска не влияет на

последующие подбрасывания. Таким образом, идеально ровная несмещенная монета являлась бы идеальным генератором случайных битовых потоков, так как значения «0» и «1» распределены равномерно и случайно. Все элементы последовательности, при таких условиях, генерируются независимо друг от друга и значение следующего элемента в последовательности не может быть предсказано, опираясь на сгенерированные ранее числа. Свойство, при котором следующее значение генератора не зависит и не является предсказуемым, исходя из предыдущего можно определить как свойство прямой непредсказуемости. Также обязательным условием хорошего датчика случайных чисел является возможность определения начального значения на основе знания каких-либо сгенерированных значений (то есть - обратная непредсказуемость). Очевидно, что использование реальных монет в методах защиты информации не представляется возможным, однако определение такого идеализированного генератора истинно случайной последовательности служит эталоном при оценке генераторов случайных и псевдослучайных чисел, а также при оценке качества выходных последовательностей алгоритмов шифрования.

Различные статистические тесты могут быть применены к последовательности для того, чтобы определить, является ли последовательность истинно случайной. Случайность является вероятностным свойством, поэтому свойства случайности последовательности принято охарактеризовать в терминах теории вероятностей. Существует бесконечное количество возможных статистических тестов, и в этой связи невозможно считать ни один конечный набор статистических тестов завершенным. Кроме этого, результаты таких тестирований следует интерпретировать с определенной долей осторожности, чтобы избежать некорректных выводов о каждом конкретном генераторе. В статистическом тестировании принято использовать понятие нулевой гипотезы: гипотезы о том, что тестируемая последовательность является случайной (H_0) и обратной ей альтернативной гипотезы H_a, которая

состоит в том, что в последовательности найдены отклонения от истинной случайности и каждый конкретный тест либо подтверждает нулевую гипотезу, либо отклоняет её. В этой связи необходимо определить возможные исходы тестирования [7] и потенциальные ошибки, представленные в таблице 1.

Таблица 1 - исходы анализа гипотез

Истинная ситуация	**Вывод**	
	Предположение H_0	**Предположение H_a**
Последовательность случайна (гипотеза H_0 верна)	Нет ошибки, предположение верно	Ошибка 1-ого рода
Последовательность неслучайна (гипотеза H_a верна)	Ошибка 2-ого рода	Нет ошибки, предположение верно

Вероятность ошибки первого рода, как правило, называют уровнем значимости теста. Эта вероятность может быть установлена до проведения эксперимента и обозначаться как α. Для теста α – это вероятность того, что тест укажет, что последовательность неслучайна, когда она действительно случайна. Вероятность ошибки второго рода принято обозначать как β. Для теста β – вероятность того, что тест укажет на то, что последовательность случайна в ситуации, когда на самом деле это не так, то есть «плохой» генератор создал последовательность, которая имеет случайные свойства. В отличие от α, β не является фиксированным значением. Параметр β может принимать много разных значений, потому что существует бесконечное число способов выявления неслучайности потока данных, и каждый из них даёт разные значения β. Вычисление ошибки второго рода является более сложным процессом, чем вычисление альфа из-за множества возможных типов неслучайности.

Одной из главных целей статистических тестов является минимизация вероятности принятия последовательности, созданной генератором как хорошей, когда на самом деле генератор был «плохим». Вероятности α и β тесно связаны друг с другом и размером выборки. Параметры размера

выборки и вероятности ошибки первого рода, как правило, выбираются при инициализации тестирования.

Каждый тест основан на вычислении тестового значения, которое является функцией данных. Статистика теста используется для вычисления P-значения, которое отражает вероятность нулевой гипотезы H_0. Для тестов это P-значение представляет собой вероятность того, что совершенный генератор случайных чисел произвел бы последовательность менее случайную, чем протестированная, учитывая тип неслучайности. Если P-значение равно 1, то последовательность признается идеально случайной и, обратно, значение P равное 0 указывает на то, что последовательность является совершенно неслучайной. Уровень значимости (α) выбирается инициатором теста. Если $P \geq \alpha$, то нулевая гипотеза принимается (последовательность случайна), в противном случае, когда $P \leq \alpha$, нулевая гипотеза отклоняется и последовательность признается неслучайной. Например, значение $\alpha = 0.01$ указывает на то, что в 1 последовательности из 100 можно ожидать отклонений, то есть значение $P \geq 0.01$ будет показывать, что последовательность считается случайной с вероятностью 99%. Значение $P < 0.01$ означает, что последовательность является неслучайной с достоверностью 99%.

Основными средствами определения степени случайности последовательности среди зарубежных методик можно выделить тесты NIST. Набор из пятнадцати статистических тестов исследует последовательности из случайных чисел с точки зрения разных критериев. Подробно тесты NIST описаны в [7].

Среди отечественных работ, связанных со статистическим тестированием, необходимо отметить Б.Я. Рябко, разработавшего несколько статистических тестов, соблюдающих высокую степень строгости. Одним из таких тестов является «стопка книг». В основе теста лежит проверка гипотезы H_0 о том, что все буквы некоторого алфавита порождаются с

равными вероятностями, против альтернативной гипотезы H_1, являющейся отрицанием H_0.

Пусть некоторый источник порождает буквы алфавита A = $\{a_1,a_2,\ldots,a_s\}$, где S>1, и требуется по выборке $x_1,x_2,\ldots,x_n$ проверить гипотезу $H_0 : p(a_1) = p(a_2) = \ldots = p(a_s) = 1/S$, против альтернативной гипотезы H_1, являющейся отрицанием H_0.

При тестировании по предлагаемому методу буквы алфавита A упорядочены (и занумерованы в соответствии с этим порядком от 1 до S), причём этот порядок меняется после анализа каждого выборочного значения x_i следующим образом: буква x_i, которую обозначаем через a, получает номер 1, номера тех букв, которые были меньше, чем номер a, увеличиваются на 1, а у остальных букв номера не меняются. Для более формального описания этого преобразования обозначим через $v^t(a)$номер буквы a принадлежит множеству A после анализа x_1, x_2 , . . . , x_{t-1} и пусть начальный порядок v^1 (•) на буквах A задан произвольно. Тогда нумерация после анализа x_t определяется следующим образом:

$$v^{t+1}(a) = \begin{cases} 1, \text{если } x_t = a, \\ v^t(a) + 1, \text{если } v^t(a) < v^t(x_t), \\ v^t(a), \text{ если } v^t(a) > v^t(x_t). \end{cases} \quad (1)$$

где v^i – номер буквы в алфавите A.

Принцип работы данного теста: как в стопке книг, если считать, что номер книги совпадает с положением в стопке. Книга извлекается и кладется наверх. Ее номер становится первым; книги, которые первоначально были над ней, сдвигаются вниз, а остальные остаются на месте.

Основная идея метода заключается в том, что подсчитывается не частота встречаемости букв в выборке $x_1,x_2,...,x_n$, а частота встречаемости номеров букв. В том случае, когда выполнена гипотеза H_1, вероятность (частота встречаемости в выборке) некоторых букв больше 1/S, и их номера в среднем будут меньше, чем у букв с меньшими вероятностями. Другими словами, книги, к которым обращаются чаще, проводят в верхней части

стопки значительно большее время, чем остальные. Следовательно, вероятность обнаружить требуемую книгу в верхней части стопки больше, чем в нижней. Если же выполнена гипотеза H_0, то очевидно, вероятность появления в выборке буквы с любым номером равна 1/S.

При применении описываемого теста множество всех {1,...,S} заранее, до анализа выборки, разбивается на r>1 непересекающихся частей A_1 = {1,2,...,k_1}, A_2 = {k_1+1,...,k_2},..., A_r = {k_{r-1}+1,...,k_r}. Затем по выборке $x_1,x_2,...,x_n$ подсчитывается количество номеров $v^t(x_t)$, принадлежащих подмножеству A_j, которое мы обозначим через n_j, j=1,...,r. При выполнении гипотезы H_0 вероятность того, что $v^t(x_i)$ принадлежит множеству A_j, пропорциональна количеству его элементов, то есть равна A_j/S. Затем по критерию χ^2 вычисляется гипотеза H_0.

$$x^2 = \sum_{i=1}^{r} \frac{(n_i - nP_i^0)^2}{nP_i^0}, \qquad (2)$$

где $P_i^0 = |A_i|/S$;

n_i– число повторений.

Ещё одним предлагаемым Б.Я. Рябко алгоритмом является «Адаптивный критерий χ2». Идея критерия заключается в следующем [75]: пусть есть алфавит A = {$a_1,a_2,\ldots,a_s$ }, где S>1 , и дана выборка, по которой надо проверить гипотезу о случайности. Имеющаяся выборка разбивается на две части, которые традиционно называются обучающей и контрольной. По первой части выборки подсчитывается частота встречаемости букв алфавита и проводится группировка (или объединение) букв с близкой частотой встречаемости в новые «супербуквы» {$A_1,A_2,\ldots,A_s$}. Затем по контрольной части выборки проверяется гипотеза H_0 о том, что вероятность встречи «супербуквы» равна $|A_i|$/S (т.е. числу букв исходного алфавита, содержащихся в подмножестве Ai, поделенному на общее число букв в исходном алфавите, что должно выполняться при равных вероятностях

встречаемости букв исходного алфавита, т.е. H_0). При больших k и некоторых отклонениях от случайности мощность данной схемы может быть существенно выше, чем у критерия χ^2, применяемого по всей выборке к исходному (несгруппированному) алфавиту.

Пусть рассматривается задача проверки основной гипотезы H_0 : $\{P(a_1) = P_1^0, P(a_2) = \ldots = P(a_s) = P_k^0$, против альтернативной гипотезы H_1, являющейся отрицанием H_0, где $\{a_1,\ldots,a_k\}$ = A- некоторое множество или алфавит, k>1. Причем для проверки гипотезы H_o против H_i имеется выборка $x_1 , \ldots , x_N$, где x_i принадлежит множеству A. Эта задача широко известна и исследована в математической статистике и существует целый ряд статистических критериев для проверки H_0 против H_1, причем одним из самых популярных и хорошо изученных является критерий Пирсона χ^2:

$$x^2 = \sum_{i=1}^{k} \frac{(v_i - \mathrm{NP_i^0})^2}{\mathrm{NP_i^0}}, \qquad (3)$$

где v_i- число встреч a_i в выборке$x_1,\ldots,x_n$;

NP – произведение длины выборки на вероятность.

В настоящей работе исследование проводится над датчиками случайных чисел, которые реализованы в современных итеративных блочных шифрах. Такая последовательность наравне с истинно случайным датчиком чисел должна быть неотличима от случайной, так как удовлетворительные статистические свойства являются необходимым условием соблюдения криптографической стойкости. В условиях невозможности получения доказательств стойкости основным способом её оценки является криптоанализ – разработка атак на криптоалгоритмы и поиск их уязвимостей. Одним из видов атак на криптоалгоритмы является атака-различитель (distinguishing attack), которая заключается в том, чтобы разработать или найти статистический критерий, проверяющий гипотезу о равномерности распределения зашифрованного текста (против её

альтернативы). Эта универсальная атака относится ко всем симметричным криптоалгоритмам. В идеале, шифр должен преобразовывать информацию так, чтобы шифрованный текст выглядел, как случайный. Однако современные шифры являются детерминированными алгоритмами, не способными генерировать истинно случайные последовательности, поэтому теоретически для любого шифра эту «неслучайность» можно обнаружить. Соответственно, важным требованием к шифру является неотличимость шифрованного текста от случайной последовательности никакими известными тестами, методами, критериями. Если шифрованный текст удается отличить от равномерно распределенных случайных чисел, то это является интересным научным результатом и указывает на недостаток шифра. Более того, подобный недостаток может быть использован для определения секретного ключа.

Как правило, криптоалгоритмы в неурезанном виде (без каких-либо упрощений) обладают достаточно хорошими свойствами (как статистическими, так и в плане стойкости) и отличить шифрованный текст от случайного за реальное время на реальных выборках невозможно, если только не использовать шифры в режиме электронной кодовой книги (Electronic Code Book, ECB), который частично сохраняет избыточность открытого текста, и построить атаку-различитель становится возможным даже не за счёт уязвимостей шифра, а за счёт избыточности языка. Для ряда задач криптографии удовлетворительные статистические свойства будут обеспечиваться за меньшее количество раундов, чем рекомендованное число.

Современные блочные шифры и криптографические хеш-функции являются итеративными, то есть они представляют собой композицию простых преобразований, называемых раундами или циклами. В среднем число раундов итеративных блочных шифров составляет от 10 до 30, хотя имеются и исключения, состоящие из меньшего или значительно большего числа раундов. Например, легковесные шифры KATAN и KTANTAN состоят из более чем 100 раундов. Понятие раунда довольно условно, поскольку он

либо может быть разбит на несколько подраундов (например, у шифров RC5 или RC6), либо, наоборот, несколько раундов могут быть сгруппированы (у шифра CAST-256 48 раундов объединены в 12 четверок). Большая часть современных шифров (даже тех, которые не играют важной роли в реальных приложениях) достаточно стойкие, и разработка атак на полноценные неурезанные версии этих шифров, как правило, невозможна, поэтому научный интерес представляют атаки на шифры с сокращенным числом раундов [8]. С увеличением числа раундов, повышается степень устойчивости шифра, но снижается скорость работы криптографического алгоритма. Таким образом, определение оптимальных характеристик работы блочных алгоритмов шифрования в совокупности с характером защищаемой информации является важной проблемой.

Проблема статистического анализа, рассмотренная выше, применительно к итеративным алгоритмам может быть расширена. При статистическом тестировании важным показателем является размер выборки (N), на котором фиксируются отклонения от случайности, причем обычно этот размер увеличивается с увеличением числа раундов шифра (r), приводя к функциональной зависимости N(r). Используя методы экстраполяции, в ряде случаев оказывается возможным спрогнозировать размер выборки на большее число раундов, когда эксперименты становятся невозможными из-за отсутствия вычислительных ресурсов для обработки большой выборки [9].

Таким образом, можно сделать вывод о том, что статистический анализ выходных последовательностей блочных шифров является актуальным способом определения степени защищенности информации обеспечиваемой этим типом криптосистем. Несмотря на то, что современные блочные криптосистемы обеспечивают высокий уровень надёжности и устойчивости к криптоанализу, не прекращается разработка новых стандартов шифрования, что подтверждается систематическим проведением конкурсов алгоритмов, а также высоким интересом криптографии в научном сообществе.

1.2 Генераторы псевдослучайных последовательностей на основе итеративных блочных шифров

На протяжении долгого времени криптография является одним из основных методов обеспечения состояния защищённости информации. Стратегической целью криптографии на протяжении многих веков неизменно является создание стойкого и удобного шифра, но нельзя утверждать, что она в полной мере достигнута. Если говорить про удобство, то проблема заключается в том, что принципы создания шифров и требования к ним существенным образом зависят от состояния технологий на момент их разработки. С одной стороны, шифры должны обеспечивать приемлемую производительность на различных программных и аппаратных архитектурах, накладывающих свои ограничения, а с другой – противостоять угрозам со стороны злоумышленников, которые могут обладать передовыми вычислительными технологиями.

Основная проблема, связанная с достижением стойкости, заключается в том, что к настоящему моменту не создан применимый на практике шифр или криптоалгоритм, для которого можно было бы строго доказать невозможность его «взлома». В частности, известный шифр Вернама, для которого получено строгое математическое доказательство его абсолютной стойкости [1], использовать на практике проблематично из-за невозможности удовлетворить требования к секретному ключу в подавляющем большинстве случаев. По этой причине на практике применяют криптоалгоритмы, для которых имеются некоторые оценки стойкости при отсутствии строгих математических доказательств. Для них существует угроза того, что рано или поздно, они окажутся успешно атакованы.

Несмотря на то, что пока еще не создан доказуемо стойкий и применимый на практике криптографический алгоритм, итеративные блочные шифры, криптографические хеш-функции и поточные шифры широко используются и, по большому счету, являются одними из наиболее надежных инструментов в системах защиты информации. Случаи, когда

утечка критически важной информации или потеря крупных финансов происходили вследствие атаки непосредственно на криптографический алгоритм, крайне редки [2]. По статистике наиболее результативные атаки используют ошибки в программных реализациях или человеческий фактор. На сегодняшний день наибольшим доверием пользуются блочные шифры и криптографические хеш-функции, что проявляется в наличии государственных стандартов на них. К числу таких алгоритмов относятся российские стандарты ГОСТ Р.34.12–2015 (на блочный шифр) и ГОСТ Р.34.11–2012 (на хеш-функцию), а также Advanced Encryption Standard (AES) – принятый в 2002 г. новый стандарт блочного шифрования США и его предшественника Data Encryption Standard (DES), активно использовавшегося с 1977 г. Высокая степень надёжности блочных шифров и криптографических хеш-функций объясняется их итеративной структурой, которая создаёт многоуровневую защиту против потенциальных атак, а также большим запасом длины ключа. Стойкость шифров подкрепляется постоянным поиском уязвимостей со стороны учёных, которых при нахождении малейшей лазейки стараются опубликовать свои результаты [9]. В итоге получают применение только алгоритмы, прошедшие интенсивные проверки учёных.

Блочными алгоритмами шифрования [4] называют симметричные системы с функцией зашифрования, реализуемой базовым блочным алгоритмом в конкретном режиме шифрования. Иными словами, блочным шифром называется симметричный криптоалгоритм, при котором весь открытый текст делится на n блоков фиксированной длины и обрабатывается поблочно. Блочные криптосистемы делятся на две основные группы: шифры перестановки (P-блоки) и шифры замены (S-блоки). Блочные шифры работают по двум основным принципам, называемым *рассеиванием* и *перемешиванием*.

Под рассеиванием следует понимать изменение любого знака открытого текста или ключа, которое повлияет на большое число знаков

получившегося шифртекста, таким образом оставляя в секрете статистические свойства открытого текста.

Под перемешиванием понимают использование преобразований, затрудняющих получение статистических зависимостей между шифрованным и открытым текстом. Основным преимуществом блочных шифров является то, что шифрование и расшифрование данных являются довольно схожими процедурами и, обычно, отличаются только последовательностью действий. Входными данными блочных шифров является блок размера n и ключ k бит. На выходе алгоритма n-битный зашифрованный блок.

Однократное применение операций рассеивания и перемешивания принято называть *раундом шифрования*. Таким образом, чтобы зашифровать открытый текст N-раундным алгоритмом, исходный текст проходит N итераций криптографического преобразования. Соответственно, столько же операций расшифрования потребуется чтобы восстановить открытый текст. Для каждого раунда шифрования из основного секретного ключа k шифра с помощью алгоритма разворачивания ключа генерируется подключ раунд k_i (где i – номер раунда). На рисунке 1 представлена одна из базовых структур блочного шифра. Алгоритмы для генерации раундовых ключей принято также называть ключевым расписанием.

Введём обозначение E – алгоритм шифрования, E^{-1} – алгоритм расшифрования. Тогда для любого фиксированного ключа функция расшифрования будет являться обратной к функции шифрования $E^{-1}_K(E_K(M)) = M$. Алгоритм шифрования будет работать тогда и только тогда, когда каждому ключу К, E_K будет являться однозначным отображением. Проще говоря, множеству битов ключа будет однозначно соответствовать множество битов шифртекста.

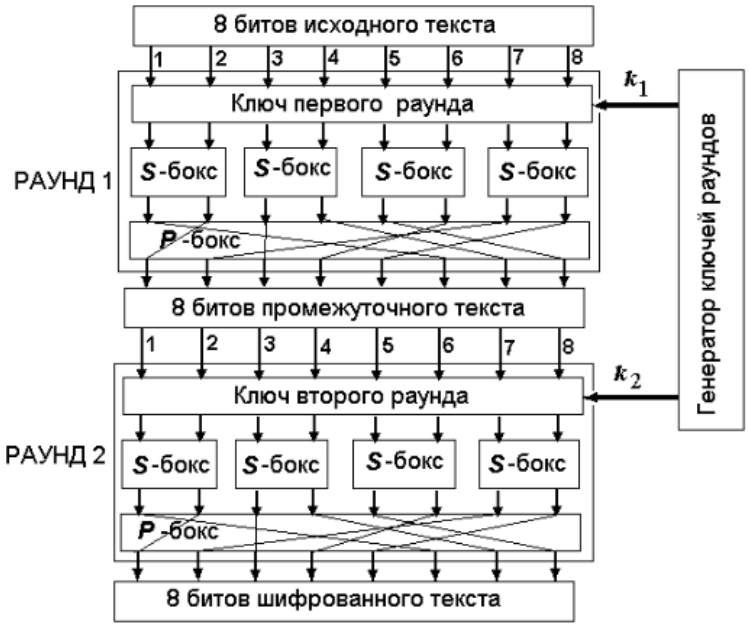


Рисунок 1 – типовая структура блочного шифра

Обязательными параметрами блочных шифров являются *размер блока*, *длина ключа* и *число раундов*. Размер блока N - это статичный параметр для каждого отдельного итеративного блочного шифра, как правило равный 64 или 128 битам. Современные шифры всё реже используют блоки в 64 бита. Размеры ключа варьируются от 40 до 256 бит. В текущих условиях вычислительных возможностей ключ в 128 бит способен выстоять перед атакой полного перебора [5].

Существует несколько режимов, таких как: ECB (электронная кодовая книга), CBC (сцепление блоков шифртекста), CFB (режим обратной связи по шифртексту), OFB (обратная связь по выходу).

В настоящей работе использован режим CTR (режим счётчика). CTR предполагает возврат на вход алгоритма блочного шифрования значение некоторого счётчика, накапливающегося с момента старта. Режим преобразует шифр в потоковый, генерируя последовательность, к которой применяется операция XOR с текстом сообщения. Исходный текст и блок зашифрованного текста имеют один и тот же размер, как основной шифр [6]. Таким образом, операцию шифрования в режиме CTR можно представить

как $C_i = P_i \oplus E_k(Ctr_i); i = 1,2, \dots m$, а операция расшифрования $P_i = C_i \oplus E_k(Ctr_i); i = 1,2, \dots m$, где (Ctr_i) – значение счётчика для i-ого блока.

1.3 Итеративные легковесные генераторы

Одну из ключевых позиций в настоящей работе занимают итеративные блочные шифры, так как именно на их примере рассмотрены генераторы псевдослучайных последовательностей. В последнее время всё чаще встречается понятие легковесных алгоритмов, предназначенных для аппаратных платформ с ограниченными вычислительными возможностями. Исследованиям легковесных шифров (lightweight cipher), использующих большое количество простых раундовых преобразований, уделяется особое внимание. Такой тип криптографических алгоритмов реализует защиту в устройствах, снабженных весьма ограниченными вычислительными ресурсами. Основной особенностью современного этапа развития Интернета и IT технологий является возрастающее количество самых различных интеллектуальных устройств, имеющих доступ в сеть Интернет. В силу условий их функционирования, а также жестких ценовых ограничений, свойственных массовому производству, эти устройства характеризуются значительными ограничениями на используемые ресурсы памяти, вычислительную мощность, источники питания и т.д. Отсюда следуют ограничения на технологии и технологические решения, предъявляемые к средствам низкоресурсной криптографии. Другой пример ограничений дают системы автоматического осуществления дорожных сборов (платы за проезд по платным дорогам): для этих систем автомобиль, двигающийся с большой скоростью, должен быть идентифицирован (authenticate) считывающим устройством на значительном расстоянии (10-12 м.) и за весьма непродолжительное время (менее 10 мс.). Ясно, что в этом случае скорость работы значительно более существены, чем размеры микросхемы или ее энергопотребление [10].

Легковесная криптография – это набор криптографических примитивов, техник и шифров, которые могут быть реализованы в очень ресурсоограниченных мобильных устройствах. Такие устройства расходуют энергию и ресурсы для всех их функций, соединяясь по каналам с ограниченной полосой пропускания и каждый элемент, используемый для безопасности, это дополнительная стоимость [11,12].

Таким образом, типичными ограничениями, встречающимися в низкоресурсной криптографии, являются: для аппаратной реализации – размер микросхемы, потребляемая энергия, время, затраченное на исполнение программы; для программной реализации – размер программного кода, размер оперативной памяти, время, затраченное на исполнение программы. Могут появляться и другие ограничения. Так, в зависимости от конкретных условий применения разрабатываемого средства, важной может оказаться такая характеристика, как ширина полосы рабочих частот канала связи. Из этого следует вывод, что проектирование подобных систем есть постоянный поиск баланса между безопасностью, ценой и производительностью. Как правило, из трёх характеристик можно выбрать только две: дёшево и быстро, но не безопасно. Или же безопасно и производительно, но дорого. На рисунке 2 представлена схема взаимосвязи характеристик криптоалгоритмов.

Рисунок 2 – характеристики легковесных итеративных блочных шифров

Наиболее активная и продуктивная деятельность по разработке низкоресурсных криптоалгоритмов происходила в области алгоритмов блочного шифрования. За последние 10 лет было предложено множество низкоресурсных решений. При этом развитие этой области легковесной криптографии шло по двум направлениям:

- эффективная реализации известных алгоритмов блочного шифрования (возможно, с их небольшой модификацией в сторону «облегчения», но при условии сохранения или незначительного снижения их криптографических свойств);
- разработка новых блочных шифров, ориентированных на оптимальную реализацию на микропрограммном или аппаратном уровне.

В целом, общей теории по разработке легковесных шифров не существует. В настоящее время разработка LWC-алгоритмов всё чаще выделяется в отдельное направление криптографии. При этом усиливается расхождение между программно- и аппаратно-ориентированными легковесными криптоалгоритмами. Так, например, итеративный блочный шифр PRESENT очень удобен для легковесной аппаратной реализации, однако требует значительных ресурсов при программной реализации. Поскольку основной задачей низкоресурсной криптографии является минимизация затрачиваемых ресурсов, важным направлением является многофункциональность – возможность с помощью одной микросхемы осуществлять шифрование, реализацию выработки имитовставки (MAC), генерацию псевдослучайной последовательности (PRNG) и т.д. При этом разрабатываемые алгоритмы должны обеспечить эффективную обработку небольших объемов данных, что наиболее характерно для встраиваемых систем [8].

Таким образом, малоресурсная криптография характеризуется следующими основными ограничениями [12]:

- размер микросхемы;
- потребляемая энергия;

- размер программного кода;
- размер оперативной памяти;
- ширина полосы рабочих частот канала связи;
- время, затраченное на исполнение программы.

В рамках данной работы рассматриваются такие легковесные шифры как PRESENT, CLEFIA и др.

Вышеописанное подчеркивает тот факт, что решая широкий спектр различных прикладных задач, криптографические алгоритмы, в том числе легковесные, могут быть использованы в различных вариациях их параметров (длины ключа, числа раундов, размера блока). Поиск рациональных комбинаций параметров работы криптографических систем является важной и современной задачей для развития систем защиты информации.

1.4 Современное применение технологий машинного обучения

История развития технологий искусственного интеллекта началась в 1950-х годах, когда во времена серьезных достижений в области информатики появились идеи передать задачи, выполняемые человеком, машине. Англоязычный термин «artificial intelligence» можно интерпретировать как «искусственная разумность». Понятие «искусственного интеллекта» можно определить как техническая или программная система, способная решать задачи, которые традиционно являются творческими, и знания о которой хранятся в памяти системы. Интеллектуальная система включает три основных блока: базу знаний, решатель и интеллектуальный интерфейс, позволяющий осуществлять коммуникации с ЭВМ без специализированных программ для ввода данных.

Этот термин часто ассоциируют с понятием «машинное обучение» и «глубокое обучение». Машинное обучение (англ. machine learning, ML) – это класс методов искусственного интеллекта, который характеризуется

следующей особенностью: не непосредственным решением задачи, а обучением в процессе применения уже готовых решений множества подобных задач [13]. В рамках машинного обучения используются алгоритмы математической статистики, численных методов, методов оптимизации, теории вероятностей, теории графов и др.

Машинное обучение можно подразделить на два основных типа:

- обучение по прецедентам, или индуктивное обучение. В основе обучения находится выявление эмпирических закономерностей в данных. Иногда методы индуктивного обучения рассматриваются как альтернатива классическим статистическим подходам, их называют «интеллектуальный анализ данных» (data mining).
- дедуктивное обучение или формализация и моделирование знаний экспертов и их ввод в информационную систему в базу знаний.

Все алгоритмы машинного обучения делят на обучение с учителем (англ. supervised learning) [14] и обучение без учителя (англ. unsupervised learning) [15]. При обучении с учителем в компьютер предварительно водятся данные, на основании которых нужно выполнить предсказание, классификацию или другие действия.

Глубокое обучение (англ. deep learning) – это разновидность машинного обучения, в основе которого лежат нейронные сети [16]. В настоящее время на использовании «глубинного обучения» построены системы распознавания речи, распознавание видеообъектов, взаимодействие компьютерных систем с естественным языком, определение смыслов в изображении. Понятие «глубина» в применении к машинному обучению обозначает моделирование многоуровненных абстракций и перевод их в данные. Соответственно, чем больше слоев имеет нейронная сеть, тем более сложные задачи могут быть ей решены.

Таким образом, задачи, которые решают средства машинного обучения, получают в последнее время всё большее распространение, так как возможности этой технологии с каждым годом возрастают и перспективы

исследований в данной области можно оценивать высоко. Несмотря на существующие российские и зарубежные статистические тесты, оценивающие псевдослучайные последовательности, разработка новых методик, основанных на методах машинного обучения, имеет важное значение для оценки степени защищенности информации.

Технологии машинного обучения в настоящее время используются в самых разных областях применения, что особенно отражается в социальной сфере жизни, а именно, в динамичном устаревании профессий и появлении новых [17]. Так, например, одна из ключевых задач настоящей диссертационной работы, связанная с машинным зрением и технологиями распознавания образов, широко применяется в медицине. В здравоохранении применение машинного обучения связано с распознаванием болезней по медицинским снимкам, анализом больших данных в медицинской сфере, поиском эффективных и малотоксичных медицинских препаратов с заранее заданными свойствами. Искусственный интеллект в этом направлении в последнее время позволил сделать качественный скачок в развитии медицинских диагностических систем, роботизированной хирургии и других сферах.

Технологии машинного обучения незаменимы в индустрии компьютерных игр. Несмотря на то, что искусственный интеллект в сфере видеоигр разрабатывается уже на протяжении десятков лет, игра в Шашки Го долгое время считалась самой сложной из классических игр для искусственного интеллекта из-за её огромного пространства вариантов ходов. Прогресс машинного обучения в данной игре был достигнут за счёт использования метода Монте-Карло для поиска в дереве и технологии обучения на основе игр машины с самой собой. Используя эти методы, компьютер выиграл 99.8% проведенных игр, в том числе обыграв чемпиона Европы в этой дисциплине во всех сыгранных партиях [18].

Машинное обучение получило широкое применение в финансовой сфере. Банки выпускают интеллектуальных виртуальных помощников,

которые на основании данных о возрасте, поле, социальном статусе и соотношении доходов и расходов дают персонализированные советы конкретному пользователю о том, как рационально распоряжаться своими средствами. Такой «помощник» по-разному взаимодействует с каждой конкретной категорией людей: система понимает, что возрастной человек готов досконально изучать информацию, а молодые люди не готовы читать длинные тексты. После того, как человек сделал крупное приобретение, система это фиксирует, и дальше, исходя из покупки, генерирует советы. Советы нужны и тогда, когда крупная покупка еще не совершена, а человек только к ней готовится. Разработка такой модели стала возможной благодаря глубокому анализу истории финансовых операций за несколько лет, при которой банк рассматривал миллиарды транзакций клиентов [19].

Особое положение в практическом применении технологий машинного обучения получила задача по распознаванию лиц, нашедшая применение в системах охраны, верификации кредитных карт, криминалистической экспертизе и других направлениях[20]. Одним из базовых алгоритмов машинного зрения, направленных на задачу распознавания лиц, является гистограмма направленных градиентов (Histogram of Oriented Gradients – HOG) [21]. Для этого модель обучается на черно-белом изображении, соседние пиксели которого сравниваются между собой. Затем добавляется стрелка, показывающая, в каком направлении изображение становится темнее. Выполнив эту процедуру для каждого отдельного пикселя изображения, каждый пиксель заменяется стрелкой. Эти стрелки называются градиентами, и они показывают направление от светлых пикселей к темным по всему изображению. Если использовать темные и светлые изображения одного человека, пиксели будут иметь различные показатели яркости, но при мониторинге направления изменений яркости, получается одинаковое изображение независимо от яркости исходной картинки. В конечном итоге исходное изображение

преобразуется так, что ясно проглядывается базовая структура лица. Чтобы найти лицо на HOG-изображении, необходимо найти часть изображения, которая наиболее похожа на известный рисунок HOG, полученный из множества других лиц в ходе обучения.

С помощью алгоритма оценки ориентиров лица (face landmark estimation) возможно оценить позицию лица тестовой фотографии. В основе этого метода преобразования изображения лежит следующий принцип: для каждой точки конечного изображения берется фиксированный набор точек исходного и интерполируется в соответствии с их взаимным положением. Аффинное преобразование является самым общим взаимно однозначным отображением плоскости на плоскость, при котором сохраняются прямые линии и отношения длин отрезков, лежащих на одной прямой. После этого преобразования лицо центрируется примерно в одно и то же положение на изображении, что делает следующий этап более точным [21].

Машинное обучение в криптографии и информационной безопасности

Актуальным аспектом развития технологий искусственного интеллекта является применение в задачах информационной безопасности, так как сетевые атаки становятся всё более сложными для определения классическими сигнатурными методами. В свою очередь, машинное обучение предлагает потенциальные решения, которые могут быть использованы для применения в сложных ситуациях, благодаря их способности быстро адаптироваться при новых и неизвестных обстоятельствах [22].

Сравнение методов обнаружения фишинга в этой области рассмотрено в работах [23-26]. Было отмечено, что многие рассматриваемые решения обнаружения фишинга из стать и [23] имеют высокую долю пропусков обнаружения. В статье [27] была предложена

автоматическая система обнаружения фишинга на основе технологий машинного обучения, используя алгоритм выбора признаков для извлечения характерных черт фишинговой электронной почты. Предлагаемый метод классификации фишинговых веб-сайтов и вредоносного программного обеспечения имел достаточно высокие показатели, около 85% корректных срабатываний.

Определенные результаты в сфере машинного обучения применительно к информационной безопасности были достигнуты в области систем обнаружения вторжений. Данные системы используются для обнаружения подозрительной сетевой активности, приводящей к нарушению основных критериев информационной безопасности: конфиденциальности, целостности и доступности. Одна из таких систем, использующих интеллектуальные технологии, была предложена в работе [28] и протестирована на данных из [29], показав высокий процент качества: всего лишь 2.01% ложных срабатываний.

В статье [30] авторы разработали алгоритм защиты от атак распределенного отказа в обслуживании (DDoS атак). С помощью системы обнаружения вторжений Snort в нейронную сеть были переданы для предобучения данные об обнаруженных вторжениях. По результатам этого исследования авторами был сделан вывод об уменьшении общего количества обрабатываемых предупреждений за счёт высокой точности классификации сетевой активности. Работа в области обнаружения вторжений, но в области беспроводных сетей, была продолжена в [31].

Нейронные сети применялись в работе [32] для изучения клавиатурного почерка: поведенческой биометрии, которая характеризует индивидуальные особенности ввода информации пользователем с клавиатуры. Нейронная сеть в данной статье с точностью 90% классифицировала пользователей по их особенностям нажатия на клавиши.

В работе [33] были предложены методы обхода систем CAPTCHA с помощью методов машинного обучения. С помощью нейронной сети [34]

производилось распознавание символов. В разных информационных сервисах средние показатели распознавания автоматически формируемых изображений для реализации теста Тюринга (различения человека от машины) составили от 5 до 65 процентов.

Одним из первых взаимодействие криптографии и машинного обучения рассмотрел в своей работе R.Rivest [35], утверждавший, что данные области знаний можно рассматривать как очень смежные и взаимозависимые.

Быстрая и эффективная криптографическая система на основе нейронной сети Хопфилда была предложена в работе [36]. А в статье [37] предлагалась методика обмена секретным ключом через общедоступный канал передачи информации с использованием нейронной сети.

С точки зрения влияния криптографии на машинное обучение, исследования в области искусственного интеллекта были сосредоточены на вопросе определения того, какие более простые классы функций могут быть изучены. Например, главный открытый вопрос в этой области заключается в том, является ли класс булевых функций отображаемым в виде булевых формул в дизъюнктивной нормальной форме, если его можно эффективно изучить на случайных примерах. Поскольку большинство отрицательных результатов в теории обучения уже зависят от криптографических допущений, отрицательные результаты в теории обучения не влияют на разработку криптографических схем. С другой стороны, положительные результаты в теории обучения, как правило, не зависят от криптографических предположений и могут в принципе применяться к криптоанализу относительно простых криптосистем.

Технологии машинного обучения уже используются в той или иной мере в задачах шифрования и сокрытия информации. Так, в работе [38] предлагается использовать интегральный классификатор, предназначенный для повышения точности методов стегоанализа, основанных на машинном обучении. Идея данного классификатора заключается в обучении нескольких

классификаторов предназначенных для распознавания пустых и заполненных контейнеров. В качестве реализации подобной концепции предлагается использовать интегральный классификатор, основанный на сжатии данных, что подразумевает выбор отдельного классификатора из набора на основе коэффициентов сжатия контейнеров. Исследователями показано, что ошибка обнаружения может быть снижена на 0.05-0.16 по сравнению с лучшими известными методами.

В работе [39] рассматривается применение технологий машинного обучения уже к задачам криптографии. Описывается известная в криптографии атака по побочным каналам (side-channel attack), а также упоминается несколько опубликованных атак на криптоалгоритмы, основанных на технологиях машинного обучения. Авторами предлагается атака на алгоритм AES, основанная на глубоком обучении, которая является более эффективной, чем существующие шаблонные атаки. В работе [40] также рассматривается атака по побочным каналам, основанная на технологиях машинного обучения, на цифровую подпись EdDSA, основанную на эллиптической кривой Эдвардса. Авторы подчеркивают, что при определенных условиях атаки, основанные на машинном обучении, могут дать абсолютную точность. Это означает абсолютную успешность атаки, при которой возможно получение единственной интерпретации открытого текста.

Криптоанализ методами машинного обучения также рассмотрен в статье Арона Гора [41], в которой итеративный алгоритм Speck подвергается дифференциальному криптоанализу. Результаты экспериментов показывают, что на данный момент средства машинного обучения не способны вытеснить традиционные методы криптоанализа, однако автор отмечает, что машинное обучение может полезно дополнить традиционные специализированные методы проведения исследований безопасности симметричных криптоалгоритмов.

Одной из наиболее близких по смыслу научных работ к настоящей диссертации можно назвать исследования, проведенные Marcello De Bernardi, Khouzani и Pasquale Malacaria [103]. В своей статье «Pseudo-Random Number Generation using Generative Adversarial Networks» учёные поставили цель определить, может ли глубокая нейронная сеть обучиться генерировать псевдослучайные последовательности так, чтобы эти последовательности отвечали требованиям безопасности. По результатам исследований, сгенерированные последовательности прошли подавляющее большинство тестов на случайность.

В целом, использование статистических методов для атаки по побочным каналам, не является новым направлением криптоанализа и в последнее время исследования в данной области доказывают эффективность данных методик, однако эти методы часто ограничены настройками размерности, ограничивающими область их применения. Использование машинного обучения при этом способно позволить сократить эти ограничения и сделать криптоанализ эффективнее [42].

В качестве базовой архитектуры нейронной сети для реализации предлагаемого в настоящей диссертационной работе метода была выбрана Inception ResNet-v2. Помимо Inception ResNet-v2 применялись современные архитектуры EfficientNet и Inception V3. Данные модели являются одними из лидеров по показателю точности классификации изображений. Конечный выбор архитектуры Inception ResNet-v2 обоснован балансом между скоростью обучения и точностью категоризации изображений, что подтверждено в статье [96], где автор применял вышеописанные модели и устанавливал зависимость между числом параметров соответствующей модели и точностью определения, при этом выполняя экспертную оценку производительности. Погрешность в рамках экспериментов настоящей работы варьировалась в рамках допустимой и существенно не повлияла на полученные результаты. Так EfficientNet-B7 показала точность категоризации на 0.05 выше Inception V3, однако в эксперименте на

контрольной выборке шифртекстов разница по точности не превосходила 1.5 процента.

Таким образом, базовой моделью для последующих экспериментов выбрана Inception ResNet-v2, базовая архитектура которой приведена на рисунке 3.

На входе выполняется свёртка 7x7 с 64 выходными каналами. Далее следует слой с максимальной свёрткой 3x3, который на выходе уменьшает высоту и ширину входного изображения в 4 раза. После этого на ветках А и Б происходит одномерная свёртка с увеличением каналов обработки изображения.

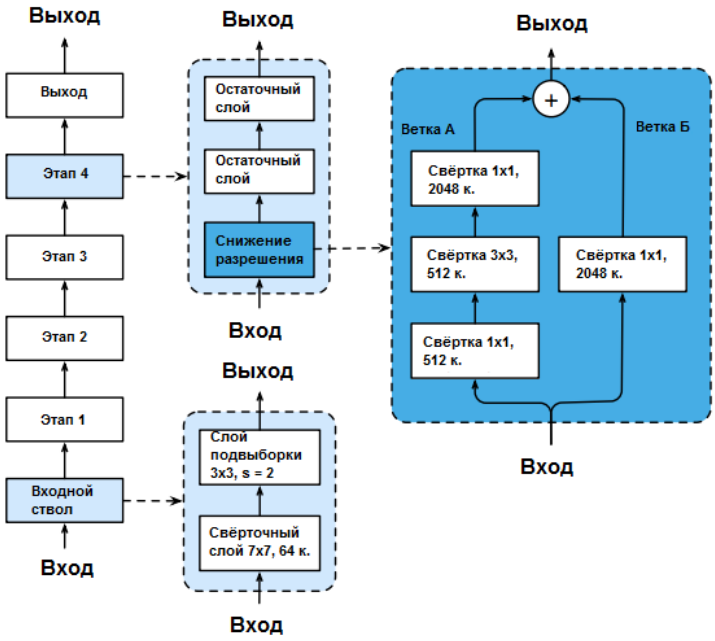


Рисунок 3 – архитектура CHC Inception ResNet-v2

Для исключения резкого снижения размерности данных используется свёрточный слой подвыборки. На последнем этапе используется несколько одномерных свёрток для наиболее точного формирования паттернов каждого шифртекста.

ВЫВОДЫ

В данной главе рассмотрены и упорядочены основные положения, касающиеся статистического анализа и математической статистики.

Представлен обзор современного состояния исследований в таких областях, как криптография, машинное обучение. Проанализированы основные научные результаты в области статистического анализа криптографических алгоритмов. Введено понятие R_{min} – минимального числа раундов, при которых выходная последовательность криптографического алгоритма неотличима от случайной.

Глава 2 посвящена построению решающих правил для псевдослучайных последовательностей с помощью классических методов: статистических тестов, использующих аппарат математической статистики.

Глава 2

Метод построения решающих правил на основе свёрточных нейронных сетей для анализа итеративных генераторов псевдослучайных чисел

В главе приводится описание метода построения решающих правил для анализа итеративных генераторов псевдослучайных чисел, основанного на применении технологий свёрточной нейронной сети «**M**achine **L**earning **S**tatistical **A**nalysis (MLSA)». Предлагаемый метод основан на задаче различения графических изображений, полученных посредством преобразования результатов работы генераторов псевдослучайных последовательностей.

2.1 Постановка задачи

Задача разработки метода построения решающих правил для анализа итеративных генераторов псевдослучайных чисел, основанного на технологиях свёрточной нейронной сети, актуальна в силу весомых перспектив технологий искусственного интеллекта. Для разработки такого метода необходимо выполнить обучение модели нейронной сети на объектах (псевдослучайных последовательностей) со статистическими свойствами различной степени случайности. Как уже было отмечено в главе 1, применение итеративных генераторов псевдослучайных чисел активно находит себя в блочных шифрах.

Идея предлагаемого метода возникла в результате наблюдения, что преобразованная в растровое изображение выходная последовательность итеративного блочного генератора, полученная на малом числе раундовых преобразований, имеет выраженную текстуру (паттерн), которая с увеличением числа раундов изменяется в сторону равномерно шумного изображения. На рисунке 4 приведён пример изменения графического

эквивалента генерируемой последовательности на примере итеративного блочного шифра Simon.

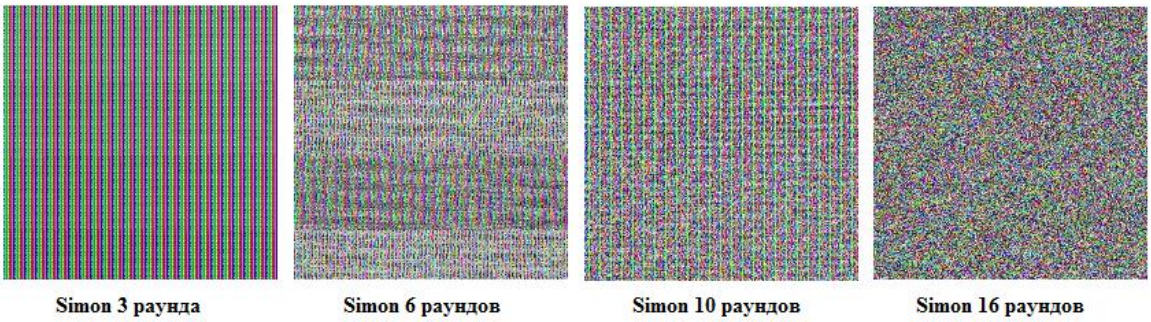


Рисунок 4 - демонстрация изменений текстуры генерируемых последовательностей на разном числе раундов (на примере итеративного блочного шифра Simon)

Из рисунка 4 видно, что последовательности на каждом из раундов обладают определенными паттернами, и эти изображения могут быть отличимы даже визуально. Согласно некоторым исследованиям, глубокие свёрточные нейронные сети, решающие задачи классификации графических изображений, превзошли результаты человека ещё в 2015[104] году и имеют очень высокие показатели точности. Основываясь на данной идее, предлагается метод построения решающих правил, основанный на применении свёрточной нейронной сети.

По результатам предварительных экспериментов, в качестве используемой модели нейронной сети была выбрана Inception ResNet-V2, обычно использующаяся для категорирования графических изображений. Модель показывает хорошие результаты, в частности в тесте ImageNet. Основным преимуществом модели Inception ResNet-V2 является высокая производительность. В работе [82] показано, что по сравнению с ранее разработанными инструментами, реализующими технологии машинного зрения, Inception ResNet-V2 существенно снизила число ошибок. Inception ResNet-V2 представляет собой нейронную сеть свёрточного типа, которая была предобучена на 1.2 миллионе изображений, разбитых на 1000 различных классов. Данная нейронная сеть состоит из свёрточных слоёв и имеет 60 миллионов параметров и 650000 нейронов.

В ходе проектирования методики статистического анализа использован подход Transfer Learning – это направление исследований в машинном обучении, при котором знания, полученные при решении одной задачи, применяются для другой. Применение такого подхода обосновано несколькими базовыми тезисами:

- выполнение обучения нейронной сети с нуля является достаточно редким явлением ввиду отсутствия необходимых датасетов;
- обучение моделей нейронных сетей с большим числом слоёв является дорогостоящим, долгим и ресурсоёмким процессом;
- при проектировании нейронной сети определить однозначно структуру, методы и параметры обучения - достаточно трудная задача, так как отсутствует какая-либо теоретическая основа для принятия подобных решений.

Transfer Learning зарекомендовал себя эффективным механизмом в задачах машинного обучения и в настоящее время является одним из ключевых элементов для исследования в прикладных задачах [88-90].

Определение окончательной архитектуры нейронной сети возможно только эмпирическим путём, подбирая различные существующие архитектуры к конкретной задаче. В настоящей работе первоначальный результат был получен посредством модели Inception V3[91].

Нейронная сеть Inception ResNet-v2n является представителем семейства современных свёрточных нейронных сетей, показывающих результат SOTA (State of the art) – наиболее продвинутые и передовые результаты в своей отрасли [92].

К семейству нейросетей, показывающих результат SOTA, можно отности следующие модели: ResNet, DenseNet, EffeceitNet, Xception [93-95]. Более новые модели нейронных сетей на эксперименте по различению цветков в статье [95] показывали лучшие результаты, чем предшественники.

Основными требованиями, предъявляемыми к задаче различения полученных последовательностей, являются:

- возможность обучения за разумное время;
- рациональное использование вычислительных мощностей (возможность работы алгоритма на удаленных платформах, а также на локальных компьютерах средней мощности);
- высокая доля точности при определении принадлежности элемента на этапе анализа контрольной выборки.

Процесс глубокого обучения, использованный в свёрточных нейронных сетях, заключается в том, что на ранних слоях определяются наиболее общие детали, тогда как последние слои занимаются уточнением специфических деталей при анализе изображения. Такой принцип работы моделей нейронных сетей можно сравнить с имитацией работы человеческого мозга, который при обучении каким-либо образам структурирует в памяти человека детали от общих к частным.

Проанализировав вышеописанные модели, был сделан вывод о том, что точность категоризации преобразованных последовательностей слабо зависит от конкретной выбранной модели: каждая из предложенных сетей показала схожие результаты, однако наиболее быстро обучалась модель Inception ResNet-v2.

Подход Transfer Learning в рамках рассматриваемой задачи заключается в предобучении последнего слоя свёрточной сети, что позволяет взять за основу базу ImageNet, но в качестве конечных объектов идентификации использовать сгенерированные на основе блочных шифров последовательности. Одно из существенных преимуществ использования реализаций нейронных сетей с помощью фреймворков заключается в том, что конечному пользователю нет необходимости углубляться в нюансы работы и, таким образом, использовать модель как «черный ящик», однако открытые исходные коды нейронных сетей позволяют выполнить модификации параметров, трансформируя типичную сборку в собственную модель, предназначенную для решения конкретных задач.

2.2 Описание универсального метода построения решающих правил MLSA

Подготовительный этап: Формирование начальной выборки

Начальный этап работы метода предполагает формирование псевдослучайных последовательностей с помощью системы «УНИБЛОКС-2015». Основной сценарий алгоритма MLSA основан на различении графических отображений соседних раундов итеративного генератора. В проведенных ранее исследованиях было установлено, что R_{min} для всех рассматриваемых генераторов превышает значение 1. Таким образом, целесообразно формировать начальную обучающую выборку для алгоритма MLSA со второго раунда до полного числа раундов R.

Практика проведения статистического анализа классическими методами показывает, что если на вход итеративного генератора подаётся случайная последовательность, то результаты статистического анализа могут быть некорректны. По этой причине в алгоритме MLSA как и в тестах из главы 3 на вход алгоритма рекомендуется подавать максимально неслучайную последовательность: в режиме счётчика возможна подача порядковых чисел или одного и того же одинакового числа. Выполняя шифрование на разном числе раундов, получаем выборку выходных последовательностей итеративного генератора со статистическими свойствами различной степени случайности. Для наглядного отображения процессов при реализации методики MLSA составлена диаграмма в соответствии с нотацией IDEF0 [101]. На рисунке 5 приведена контекстная диаграмма верхнего уровня для процесса проведения статистического анализа алгоритмов методом MLSA.

Рисунок 5 - контекстная диаграмма верхнего уровня

Приведенная контекстная диаграмма верхнего уровня отражает входные и выходные данные процесса, а также используемые элементы управления (MLSA) и механизмы: библиотеку «УНИБЛОКС-2015», модель Inception ResNet-V2, утилиту для конвертации шифртекстов в графические изображения TextToBMP Utility, а также инструментарий электронных таблиц. Контекстная диаграмма верхнего уровня декомпозируется на процессы формирования начальной выборки (рисунок 5). На рисунке 6 представлена диаграмма A1, являющаяся декомпозицией диаграммы верхнего уровня и отражающая основные процессы метода MLSA.

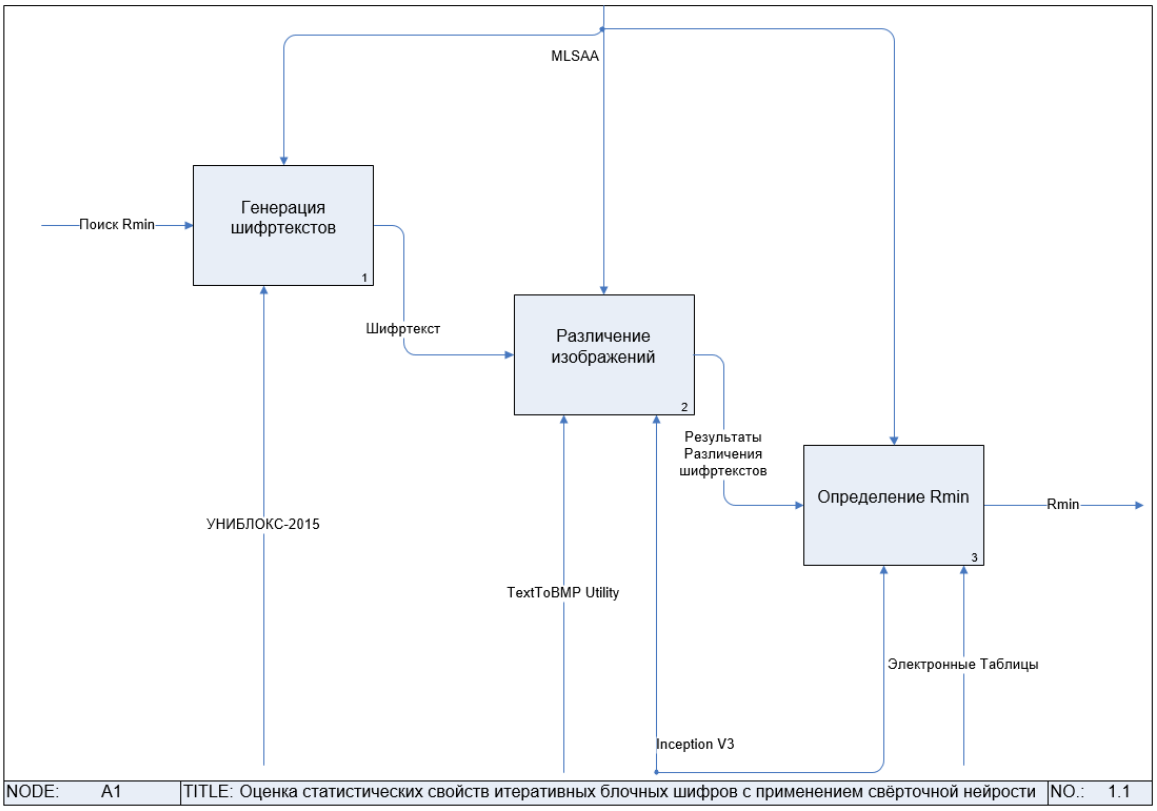


Рисунок 6 - Диаграмма A1

На диаграмме A1 представлен ход процессов работы метода MLSA. Диаграмма A1.1 декомпозирует процесс формирования начальной выборки шифртекстов для их последующего анализа методом MLSA (рисунок 7).

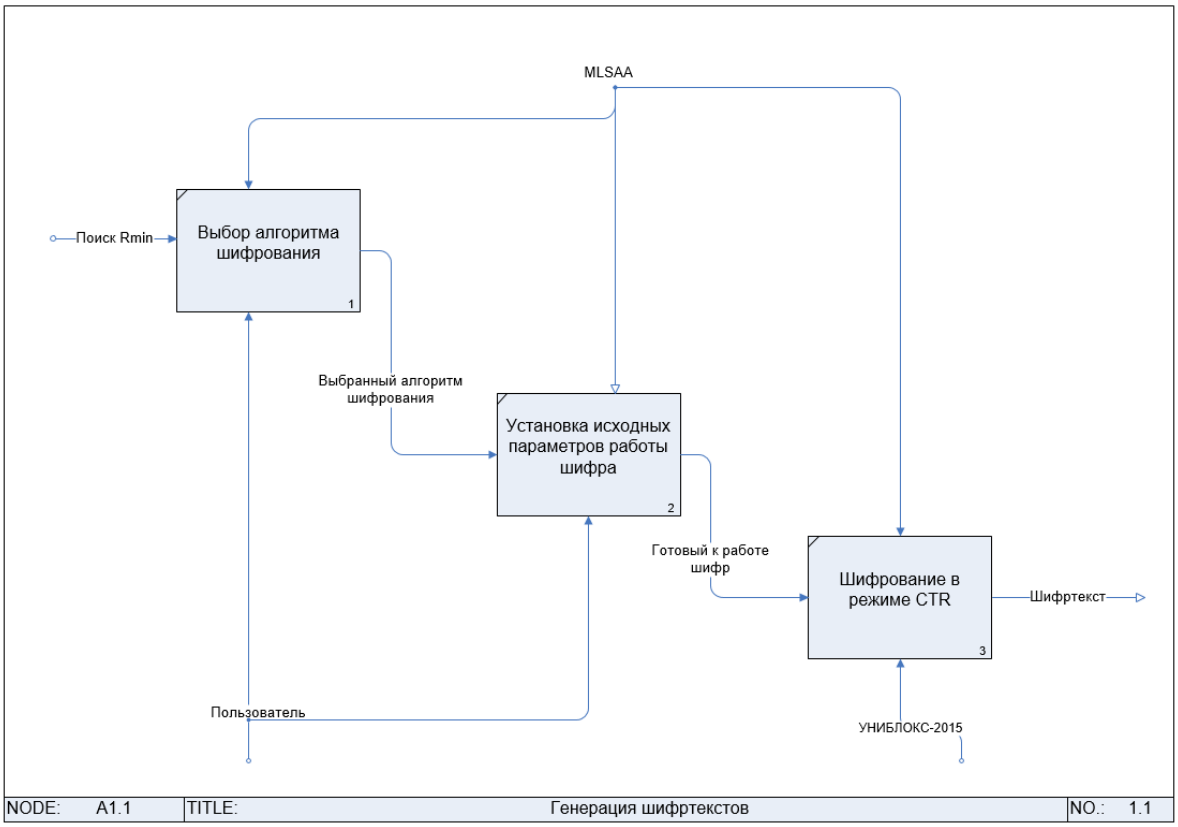


Рисунок 7 - Диаграмма A1.1.

Последующие декомпозиции процессов работы метода MLSA будут представлены в следующих разделах.

Адаптация сгенерированных последовательностей к предлагаемому алгоритму.

Сформированные генератором последовательности должны быть преобразованы в формат растровых графических изображений для того, чтобы выбранная модель Inception ResNet-v2 корректно выполнила процесс обучения. Преобразование шифртекстов в графический формат (альтернативно назовём графические эквиваленты выходных последовательностей блочных шифров – *отображениями шифртекстов*) производится автоматизировано с помощью разработанной программной утилиты на языке C++. Исходный код предлагаемой программы-преобразователя находится в Приложении И.

Утилита выполняет конвертацию текстового файла (или опционально - потоковой последовательности) в BMP файл с расширением 24 бита, где каждая компонента палитры RGB кодируется 8 битами (палитра содержит 256 цветов). Принцип преобразования шифртекстов в отображения заключается в последовательном присваивании компонентам палитры RGB значений считываемых байт из зашифрованной последовательности.

Ниже приводится пример верификации корректности реализации программной утилиты-преобразователя. На рисунках 1-3 приведены примеры конвертации: на рисунке 6 используется систематически повторяющийся символ, рисунок 8 получен получен из повторяющейся последовательности символов (1234qwer1234qwer…), на рисунке 9 представлен результат конвертации в изображение, являющееся результатом работы генератора rand() языка C++ соответственно.

Рис. 8 Результат преобразования

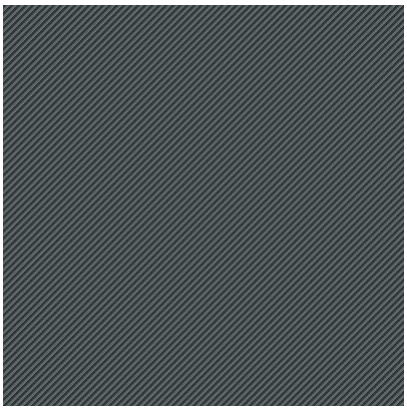


Рис. 9 Результат преобразования

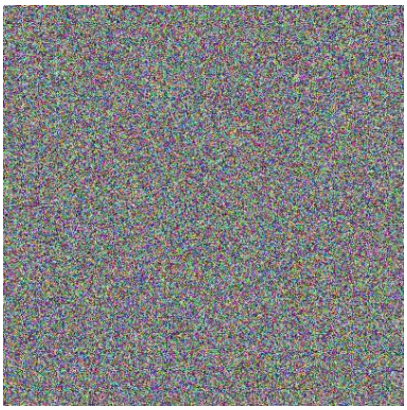


Рис. 10 Результат преобразования

Диаграмма A1.2 нотации IDEF0 отражает непосредственный процесс работы с графическими представлениями сгенерированных последовательностей (рисунок 11).

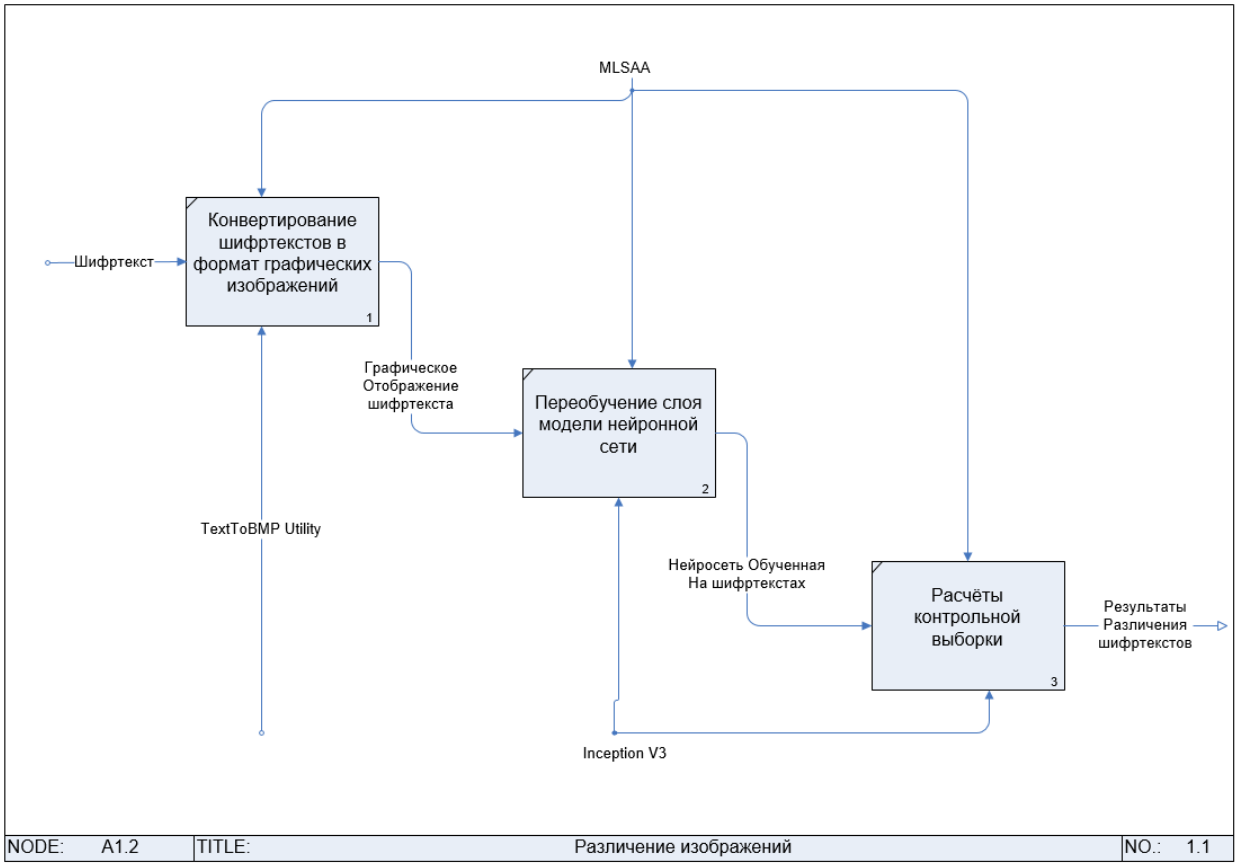


Рисунок 11 - Диаграмма A1.2

После преобразования полученных последовательностей с помощью разработанной на языке C++ утилиты TxtToBMP Utility и формирования выборки уже на уровне изображений происходит переобучение последнего слоя модели на графические отображения последовательностей, после чего происходит различение соседних раундов работы криптографического итеративного генератора, описывающееся в следующем разделе.

Предлагаемый метод имеет два основных сценария. Общая схема работы представлена на рисунке 3. *Сценарий анализа, основанный на сравнении соседних раундов* одного и того же алгоритма, предполагает обучение нейронной сети на полученных последовательностях каждого из раундов до R, где R - полное число раундов.

Эталонный сценарий выполняет аналогичные сравнения последовательностей, однако выборка на каждом раунде сравнивается с генератором на полном числе раундов (в экспериментах предлагалось

сравнение с полнораундовым шифром AES, использующим 14 раундов для 256-битного ключа).

Для тестирования по предлагаемому методу необходимо выполнить обучение СНС. Ниже представлен псевдокод *алгоритма 1*, описывающий процесс обучения СНС.

Алгоритм 1. Обучение свёрточной нейронной сети

Шаг 1: **Функция** ОбучитьНейроннуюСеть (*Cipher*, *r*, *M*)
Cipher – итеративный блочный генератор;
r – число раундов генератора;
M – размер обучающего множества;
Шаг 2: Сгенерировать *M* выборок с помощью генератора Cipher и получить $\tilde{y}^r = (\tilde{y}_1^r, \dots, \tilde{y}_M^r)$
Шаг 3: **Если** выбран «эталонный сценарий», **то** Сгенерировать *M* выборок с помощью генератор*а AES14* и получить $\tilde{y}^{rand} = (\tilde{y}_1^{rand}, \dots, \tilde{y}_M^{rand})$
Иначе если выбран сценарий «соседних раундов» **то** Сгенерировать *M* выборок с помощью генератора Cipher и получить $\tilde{y}^{r+1} = (\tilde{y}_1^{r+1}, \dots, \tilde{y}_M^{r+1})$
Шаг 4: Преобразовать сгенерированные множества выборок $\tilde{y}^r$ **и** ($\tilde{y}^{rand}$**или** $\tilde{y}^{r+1}$) в изображения. Полученные множества обозначим $y^r = (y_1^r, \dots, y_M^r)$, $y^{rand} = (y_1^{rand}, \dots, y_M^{rand})$, $y^{r+1} = (y_1^{r+1}, \dots, y_M^{r+1})$
Шаг 5: Обучить нейронную сеть различать изображения из обучающих выборок y^r **и** (y^{rand} **или** y^{r+1}).
Шаг 6: **Вернуть** НейроннаяСеть^r(image), которая будет относить image к 0 (случайному) или 1 (неслучайному) результату генерации.

Практика проведения статистического анализа классическими методами показывает, что если на вход итеративного генератора подаётся случайная последовательность, то результаты статистического анализа могут быть некорректны. По этой причине в методе MLSA, как и в тестах из главы 3, на вход алгоритма рекомендуется подавать максимально неслучайную последовательность: в режиме счётчика возможна подача порядковых чисел или одного и того же одинакового числа. Выполняя шифрование на разном числе раундов, получаем выборку выходных последовательностей итеративных генераторов со статистическими свойствами различной степени

случайности. На *шаге 2-3* приведённого алгоритма 1 с помощью системы «УНИБЛОКС-2015» выполняется генерация последовательности.

На *шаге 4* сформированные последовательности преобразуются в формат растровых графических изображений для того чтобы нейронная сеть корректно выполнила процесс обучения. Преобразование сгенерированных последовательностей в графический формат (альтернативно назовём графические эквиваленты выходных последовательностей итеративных генераторов – отображениями) производится автоматизировано с помощью разработанной программной утилиты на языке C++. Утилита выполняет конвертацию текстового файла (или опционально – потоковой последовательности) в BMP файл с расширением 24 бита, где каждая компонента палитры RGB кодируется 8 битами (палитра содержит 256 цветов). Принцип преобразования полученных последовательностей в изображения заключается в последовательном присваивании компонентам палитры RGB значений считываемых байт из зашифрованной последовательности.

После преобразования сгенерированных последовательностей с помощью разработанной на языке C++ утилиты TxtToIMG Utility и формирования выборки уже на уровне изображений происходит переобучение последнего слоя свёрточной нейронной сети на графические отображения (*шаг 5 алгоритма 1*). В процессе обучения нейронная сеть запоминает основные паттерны, характерные для отображений на разном числе раундов, которые использует для последующей категоризации.

После выполнения процесса обучения СНС выполняется распознавание псевдослучайных последовательностей. В зависимости от выбранного сценария на контрольной выборке СНС сравнивает шифртексты на соседних раундах шифрования (или сравнивает шифртексты выбранного раунда с «эталоном», например полнораундовым блочным шифром AES), подсчитывает процент верных решений при определении принадлежности элемента контрольной выборки к тому или иному множеству. С увеличением

числа раундов шифрования и, соответственно, улучшением статистических свойств, модель увеличивает значение E (E – число ошибок, допущенных моделью при определении принадлежности).

Алгоритм 2. Статистический тест определения случайности выборки методом MLSA

Шаг 1: **Функция** РаспознатьПоследовательность (*x*, *Cipher*, *r*)
x – запрошенная на генерирующем устройстве выборка (генерируется в режиме CTR в сценарии chosen-plaintext attack);
Cipher – итеративный генератор;
r – число раундов генератора;
Шаг 2: Выбрать размер обучающей выборки *M*
Шаг 3: НейроннаяСеть^{r}(image) = ОбучитьНейроннуюСеть (*Cipher*, *r*, *M*)
Шаг 4: Представить выборку *x* в виде изображения *image*
Шаг 5: *Result*:=НейроннаяСеть^{r}(*image*)
Шаг 6: **Если** Result = 0, **то вернуть** «выборка случайна»
иначе если Result = 1, **то вернуть** «выборка неслучайна»

Сценарий соседних раундов метода MLSA

Сценарий анализа, основанный на сравнении соседних раундов одного и того же алгоритма, предполагает обучение на псевдослучайных последовательностях каждого из раундов до R, где R – полное число раундов.

На контрольной выборке модель сравнивает последовательности на соседних раундах и подсчитывает процент верных решений при определении принадлежности элемента контрольной выборки к той или иной группе (т.е. раунду). С увеличением числа раундов работы генератора и, соответственно, улучшением статистических свойств, модель увеличивает значение E (где E – число ошибок, допущенных моделью при определении принадлежности).

Процесс формирования конечных результатов проведения эксперимента представлен на рисунке 12.

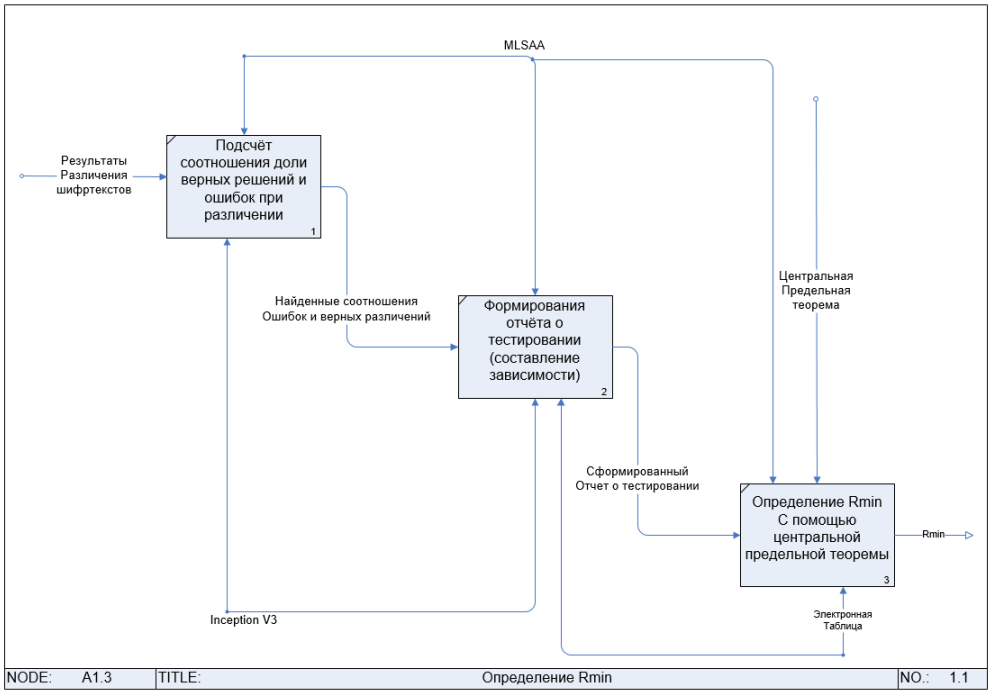


Рисунок 12 – Диаграмма A1.3

Как было описано выше, после выполнения эксперимента на контрольной выборке формируется соотношение ошибок и верных решений нейронной сети.

Эталонный сценарий метода MLSA

Эталонный сценарий предполагает наличие последовательности, которая обладает удовлетворительными статистическими свойствами. В качестве такой последовательности может использоваться исследуемый алгоритм на полном числе R. В качестве альтернативы может быть использован результат работы генератора AES на полном числе R (12-14 раундов преобразования). С помощью обоих сценариев возможен поиск значения параметра R_{min} – минимального числа раундов генератора, при котором обеспечиваются удовлетворительные статистические свойства. Алгоритм 3 описывает процесс поиска R_{min}.

Алгоритм 3. Определение параметра R_{min} методом MLSA

Шаг 1: **Функция** Вычислить R_{min} (*Cipher*)
Шаг 2: Для всех r = $\overline{1, R}$

НейроннаяСеть^r = ОбучитьНейроннуюСеть (*Cipher*, *r*, *M*)
Шаг 3: $x^r = (x_1^r, \ldots, x_N^r)$
Шаг 4: **Пока** ($E_0 + E_1 \in [M - \delta_\alpha, M + \delta_\alpha]$)
Выполнять ($E_0 + E_1$) := ВычислитьОшибку(*Cipher*)
r:=r+1
Шаг 5: **Вернуть** r

Подсчитываются значения $S'^r_n = S^r_n/n$ – доля выборок, удовлетворяющих критерию случайности, таких, что $S'^r_n \in [0,1]$. Определение случайности выборки получено из определения интервала ошибки нейронной сети, который определяется с помощью центральной предельной теоремы.

Пусть алгоритм шифрования для тестирования выбран, тогда сформируем x_i^r , где r = $\overline{1,R}$ – выборка, полученная посредством применения шифра с r раундами шифрования в режиме CTR.

Пусть $\xi_i^r = \begin{cases} 0 \ - \text{ выборка (изображение)} x_i^r \text{ признана случайной} \\ 1 \ - \text{ выборка (изображение)} x_i^r \text{ признана неслучайной} \end{cases}$, тогда $S_n^r = \sum_{i=1}^N \xi_i^r$ – количество выборок признанных случайными (r = $\overline{1,R}$ – раунд шифрования; i = $\overline{1,N}$, где N – число выборок). $X_n^r = (x_1^r, .., x_N^r)$ – контрольные выборки шифртекстов. Обозначим $S'^r_n = S^r_n/n$ – доля выборок, признанных случайной, таких, что $S'^r_n \in [0,1]$. Когда доля признанных случайными выборок $S'^r_n \in [M - \delta; M + \delta]$, фиксируется значение r = R_{min}.

$$\boldsymbol{R_{min} = min\left\{ r \geq 0 \,\middle|\, S'^r_n \in \left[\frac{1}{2} - \delta_\alpha, \frac{1}{2} + \delta_\alpha\right]\right\}}$$

Основным преимуществом предлагаемого метода является возможность использования выборки меньшей, чем требуется для классических статистических тестов. Экспериментальное обоснование эффективности предлагаемого метода приведено в главе 3. За счёт подхода нейронных сетей к классификации графических изображений, при котором анализируется не каждое текущее значение счётчика, как при тестировании классическими методами, а определяются общие паттерны всей выборки, её объем может не превышать $2^{21.9}$ бит информации.

Сценарий анализа одинаковых раундов различных алгоритмов

При проведении анализа по данному сценарию алгоритм обучается на выборках всех раундов до R на последовательностях двух и более итеративных генераторов. Сценарий является альтернативой сценарию «соседних раундов». Контрольная выборка выполняет различение одинаковых раундов разных алгоритмов. Экспериментально было показано, что на ранних раундах модель нейронной сети с высокой вероятностью может отличить выходную последовательность одного алгоритма от другого. По аналогии с вышеописанным сценарием, R_{min} достигается при доле ошибок модели, стремящейся к 0.5. В таблице 1 приведён пример различения алгоритмов Speck и Present. В данном случае модель снова получает наиболее сложные условия работы, так как оба генератора при статистическом анализе средствами тестов NIST демонстрируют схожее поведение (примерно эквивалентное прохождение статистических тестов на каждом раунде и эквивалентный Rmin).

Таблица 2. Эксперимент по сценарию 2.

R	3	4	5	6	7	8	9	10
%	100	100	99	100	90.9	65.3	59	48.6

Необходимо отметить, что данный сценарий имеет существенную уязвимость в виде необходимости выбора генератора-оппонента со схожим поведением, в противном случае результаты тестирования могут не быть абсолютно объективными. В представленном примере работы сценария алгоритмы Speck и Present демонстрируют схожее поведение при статистическом анализе с помощью тестов NIST (Рисунки 13 и 14).

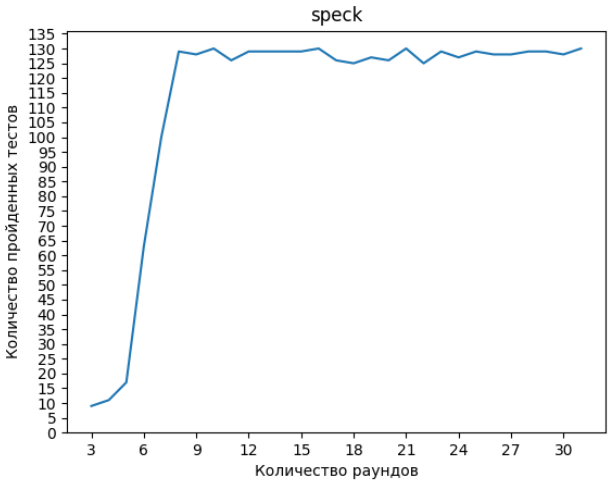


Рисунок 13 – график NIST Speck

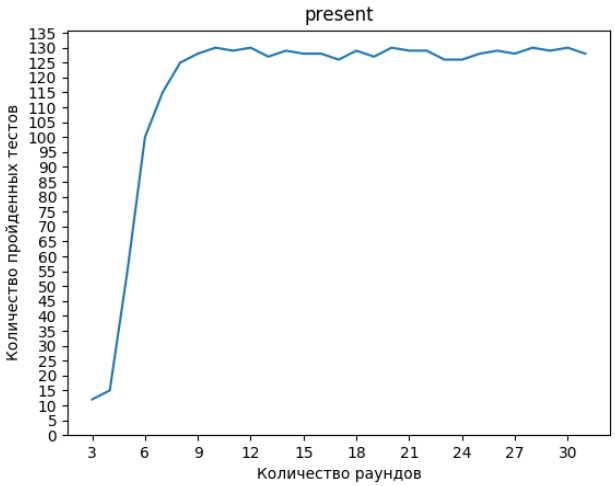


Рисунок 14 – график NIST Present

Корректный выбор алгоритмов-оппонентов позволил получить результаты, сопоставимые с результатами, полученными с помощью группы тестов NIST, однако необходимо отметить, что в случае с этими алгоритмами группа тестов NIST являлась самым строгим инструментом определения качества выходной последовательности.

Сценарий основанный на классических тестах

Идея данного сценария заключается в том, что обучающая выборка формируется на основании данных, полученных в главе 2. Выборка псевдослучайных последовательностей с удовлетворительными статистическими свойствами при таком сценарии может быть сформирована из последовательностей, зашифрованных на раундах $>R_{min}$. Альтернатива – подавать в качестве обучающей выборки таких последовательностей результат преобразования на полном числе раундов разных алгоритмов либо результат преобразования на полном числе раундов какого-либо алгоритма с заведомо удовлетворительными статистическими свойствами (например, выходную последовательность алгоритма AES на полном числе раундов очевидно фактически обладает последовательностью, неотличимой от случайной). В отличие от предлагаемых ранее сценариев, данная методика не

использует различение раундов между собой. Используются уже заготовленные обучающие выборки, основанные на данных, полученных статистическими тестами.

В качестве противоположной выборки подаются последовательности, преобразованные на числе раундов меньшем, чем R_{min}. Этот метод показывает высокие результаты и на обучающей выборке чуть более, чем 2^{14} (по 2^{13} последовательностей с удовлетворительными и неудовлетворительными статистическими свойствами соответственно) последовательностей и контрольной выборке 2^{11} в 98% случаев верно приняла решения относительно принадлежности последовательности к той или иной группе.

У предлагаемого сценария есть существенный недостаток, заключающийся в зависимости формирования выборок от заведомо полученных результатов тестирования, так как поиск границы R_{min} при использовании разных статистических тестов, а также при различии входных данных о тестировании, может давать отличные друг от друга результаты, что ставит объективность полученного результата под сомнение, однако применение данного сценария тоже возможно.

2.3 Теоретическое обоснование метода MLSA

Рассматриваемые сценарии 1 и 2 предполагают нахождение значения R_{min}, прибегая к понятию доверительного интервала, который можно найти с помощью центральной предельной теоремы. Пусть ξ - потенциально возможный исход тестирования, где 1 – модель приняла верное решение, 0 – неверное, тогда при уровне доверия $\alpha = 0.99$ найдем интервал δ с помощью центральной предельной теоремы.

Теорема. Пусть n – размер контрольной выборки и α - уровень значимости решающего правила, тогда при $\tilde{\delta}(n, \alpha) = \frac{Q_{\alpha/2}}{2\sqrt{n}}$, где

$\boldsymbol{Q_{\alpha/2} = F_{0,1}^{-1}\left(1-\frac{\alpha}{2}\right)}$ – **к**вантиль стандартного нормального распределения уровня $\boldsymbol{1-\frac{\alpha}{2}}$, выполняется

$$\boldsymbol{\tilde{S}_n \subset [{}^1\!/_2 - \tilde{\delta}(n,\alpha); {}^1\!/_2 + \tilde{\delta}(n,\alpha)]} \leq \alpha.$$

Доказательство. Введем следующий случайные величины для $i = \overline{1,n}$.

$$\eta_i = \begin{cases} 1 - \text{нейронная сеть приняла верное решение на } i\text{ом изображении} \\ 0 - \text{нейронная сеть ошиблась} \end{cases},$$

Тогда количество верных решений нейронной сети ($\boldsymbol{S_n}$) и их долю ($\boldsymbol{\tilde{S}_n}$) на всей контрольной выборке можно определить следующим образом:

$$\boldsymbol{S_n = \sum_{i=1}^{n} \eta_i}, \boldsymbol{\tilde{S}_n = S_n / n}.$$

Найдем теперь такое $\tilde{\delta}(n, \alpha)$, что при $\boldsymbol{\tilde{S}_n \subset [{}^1\!/_2 - \tilde{\delta}(n,\alpha); {}^1\!/_2 + \tilde{\delta}(n,\alpha)]}$ можно было бы сделать вывод о том, что нейронная сеть способна отличать сгенерированные и случайные последовательности друг от друга эффективнее простого угадывания. Пусть

$$\boldsymbol{S_n^* = \frac{S_n - ES_n}{\sqrt{nD\eta_1}}}, \quad (1)$$

где ES_n и DS_n – соответственно математическое ожидание и дисперсия S_n.

Из центральной предельной теоремы следует, что $P\,(S_n^* \subset [-\delta; \delta]) \approx F_{0,1}(\delta) - F_{0,1}(-\delta)$, где $F_{0,1}(\cdot)$ – функция стандартного нормального (гауссовского) распределения.

Пусть $Q_{\alpha/2} = F_{0,1}^{-1}\left(1-\frac{\alpha}{2}\right)$ – квантиль стандартного нормального распределения уровня $1-\frac{\alpha}{2}$, тогда $P\left(S_n^* \subset \left[-Q_{\alpha/2}; Q_{\alpha/2}\right]\right) \approx 1 - \alpha$.

Если нейронная сеть неспособна отличать графические эквиваленты сгенерированных и неотличимых от случайных последовательностей друг от друга, то все случайные величины η_i будут иметь распределение Бернулли с параметром 1/2, т.е. $P(\eta_i = 1) = P(\eta_i = 0) = 1/2$, поскольку результат работы нейронной сети будет равносилен случайному угадыванию.

Следовательно, ES_n = n/2, DS_n = n/4, а формулу (1) можно преобразовать к виду

$$S_n^* = \frac{S_n - n/2}{\sqrt{n}/2} = \frac{2S_n - n}{\sqrt{n}}. \quad (2)$$

Далее из (2) получаем, что $\tilde{\delta}(n, \alpha) = \frac{Q_{\alpha/2}}{2\sqrt{n}}$. **Теорема доказана.**

Например, при $\alpha = 0.01$ величина $Q_{0.01/2}$ равна 2.59, и при таких значениях $\tilde{\delta}(0.01; 200) = 0.09$, и $\tilde{\delta}(0.01; 2000) = 0.03$

Таким образом, при построении решающих правил методом предлагаемым методом MLSA (Machine learning statistical analysis) попадание доли ошибки в найденный интервал свидетельствует о достижении генератором статистических свойств, при которых последовательность неотличима от случайной.

Основным преимуществом предлагаемого метода является возможность использования выборки меньшей, чем требуется для классических статистических тестов. Экспериментальное обоснование эффективности предлагаемого метода приведено в главе 3. За счёт подхода нейронных сетей к классификации графических изображений, при котором анализируется не каждое текущее значение счётчика как при тестировании классическими методами, а определяются общие паттерны всей выборки, её объем может не превышать $2^{21.9}$ бит информации.

ВЫВОДЫ

В главе 2 предложен метод построения решающих правил MLSA, в основе которого лежат технологии машинного обучения, а именно технологии распознавания изображений. Рассмотрены несколько сценариев проведения статистического анализа с помощью предлагаемого метода MLSA. Приведены алгоритмы по обучению СНС, статистическому тестированию и поиску минимального числа раундов шифрования.

В главе 3 будут приведены эксперименты по предлагаемому методу, практические обоснования эффективности и результаты проведенных исследований. Результаты главы отражены в [97-100].

Глава 3

Статистическое тестирование итеративных генераторов на примере блочных шифров

В главе описаны особенности разработанной информационно-аналитической системы. Приведены основные приёмы программирования и результаты тестирования алгоритмов с помощью предлагаемого в диссертационной работе метода, а также с помощью классических статистических тестов.

3.1 Постановка задачи

Как уже было отмечено в главе 1, обеспечение удовлетворительных статистических свойств итеративными генераторами является необходимым условием криптографической стойкости блочных шифров. Таким образом, исследование зависимостей качества статистических свойств от числа раундов является актуальной задачей, однако их исследование сопряжено с рядом проблем ввиду необходимости специализированного программного средства.

Необходимость разработки информационной системы для анализа статистических свойств итеративных блочных шифров обусловлена отсутствием подобного универсального средства тестирования алгоритмов, которое бы имело в своём составе набор отлаженных шифров, имеющих единый интерфейс. Наличие большого количества открытых исходных кодов и криптографических библиотек от разных разработчиков не решает эту проблему.

Одним из вариантов использования итеративных блочных шифров является генерация псевдослучайных чисел в режиме счетчика (CTR), подразумевающим последовательное шифрование значений некоторого счетчика и формирование псевдослучайной последовательности из выходных

блоков или их частей. Удовлетворительные статистические свойства выходной псевдослучайной последовательности могут быть обеспечены значительно меньшим числом раундов, чем полное число раундов генератора. Очевидно, что сокращение числа раундов увеличит производительность алгоритмов и позволит генерировать псевдослучайные числа быстрее, и, даже если такой усеченный генератор будет иметь высоковероятные характеристики (например, дифференциальные, линейные или интегральные), он, тем не менее, сможет генерировать псевдослучайные последовательности с удовлетворительными статистическими свойствами, поскольку вероятность появления блоков с требуемыми на входе шифра свойствами может быть ничтожно малой [43].

Современные итеративные блочные шифры помимо секретного ключа используют, так называемые, раундовые ключи. Принимая во внимание основное правило криптографии – принцип Керкгоффса [44], который гласит о том, что за конфиденциальность выходной последовательности отвечает только ключ, можно сделать вывод о том, что задача генерации качественного ключа является одной из важных. Чтобы сделать вывод о работе генератора псевдослучайных чисел, а также оценить степень случайности выходной последовательности итеративного блочного шифра, необходимо провести тестирование статистических характеристик. Необходимость в генерации качественной псевдослучайной последовательности, а также соблюдении хороших статистических свойств обусловлена тем, что ни результат преобразования, ни ключевая последовательность не должны быть предсказуемы для злоумышленника. Имея информацию о статистических уязвимостях ключа алгоритма, криптоаналитик может существенно снизить диапазон перебираемых ключей, что позволит реализовать атаку полным перебором. В ситуации, когда злоумышленник перехватывает зашифрованное сообщение, неудовлетворительные статистические свойства могут послужить причиной атаки по известному шифртексту.

Следует отметить, что задачи, которые решают криптографические алгоритмы, сводятся к получению последовательности из двоичных символов, а не байт. В данном случае качественный генератор можно сравнить с идеально ровной монетой, вероятность выпадения каждой из сторон которой строго равна 0.5. Генераторы случайных чисел можно поделить на два основных типа: истинно случайные-физические генераторы/датчики случайных чисел и псевдослучайные-программные датчики/генераторы случайных чисел. Первые принимают на вход некий случайный бесконечный процесс, а на выходе дают бесконечную (зависит от времени наблюдения) последовательность 0 и 1. Вторые представляют собой заданную разработчиком детерминированную функцию, которая инициализируется так называемым зерном, после чего также на выходе выдает последовательность 0 и 1. Зная это зерно, можно предсказать всю последовательность. Хороший программный датчик случайных чисел — это тот, для которого невозможно предсказать последующие значения, имея всю историю предыдущих значений, то есть, не имея зерна. Задача восстановления предыдущего члена последовательности, заключающаяся в определении элемента последовательности ai-1 по известным k членам последовательности $a_i a_{i+1} a_{i+2} ... a_{i+k-1}$, называется непредсказуемостью влево. Существует и обратная задача предсказания следующего члена последовательности, при которой по известным k членам последовательности $a_{i-k+1} a_{i-k+2} ... a_{i-1} ai$ предсказывается значение a_{i+1} называется непредсказуемостью вправо [45].

Использование истинно случайных (физических) датчиков случайных чисел сопряжено с рядом проблем:

- Случайное явление/процесс, которое берется за основу, может быть не способно выдавать числа с нужной скоростью.
- Степень случайности некоторых физических явлений можно поставить под сомнение. Например, электромагнитный шум может быть суперпозицией нескольких более-менее однообразных периодических

сигналов.

Таким образом, поскольку блочные шифры претендуют на универсальность, решая целый спектр прикладных задач (генерация псевдослучайных последовательностей и чисел является одной из них), то при создании новых алгоритмов, помимо числа раундов, обеспечивающего криптографическую стойкость, имеет смысл указывать и число раундов, обеспечивающее удовлетворительные статистические свойства, не гарантируя криптографической стойкости [3]. Следует отметить, что ряд итеративных блочных шифров уже сейчас имеет варьируемые рекомендуемые параметры: например, разработчики шифра AES определяют рекомендуемое число раундов от 10 до 14 в зависимости от длины ключа, которая также изменяется от 128 до 256 бит.

Необходимым условием определения такого числа раундов является статистический анализ блочных шифров. При проведении статистического анализа зачастую исследуются свойства шифра в зависимости от числа раундов (как правило с ростом числа раундов статистические свойства шифров улучшаются). Для проведения такого анализа целесообразно использовать программные коды шифров, имеющиеся в открытом доступе, однако интеграция этих кодов в собственные программы для статистического тестирования сопряжена с рядом проблем, которые вызваны, главным образом, различными сигнатурами (программными интерфейсами) функций шифрования и развертывания ключа. Две наиболее существенные проблемы, затрудняющие автоматизированную обработку шифров, следующие: во-первых, число раундов не выносится в качестве аргумента функции и, во-вторых, блоки представляются в виде слов разной длины (например, 128-битовый блок может быть представлен как четыре 32-битовых слова или как шестнадцать 8-битовых).

Разработанная на базе открытых исходных кодов шифров программная библиотека с классическими статистическими тестами, где реализации шифров обладают унифицированным интерфейсом, позволяет проводить их

автоматизированное тестирование.

3.2 Тестирование итеративных генераторов с помощью метода MLSA

Для проведения статистического анализа предлагаемым методом необходимо, чтобы свёрточная нейронная сеть с высокой точностью выполняла распознавание сгенерированных последовательностей, сохраняя при этом высокую скорость обучения. Хотя применяемые в работе архитектуры обладают одними из наиболее высоких показателей точности классификации изображений, выбор исходных параметров нейронной сети является важной задачей.

Исходя из вышеописанных требований, помимо выбора конкретной модели машинного обучения необходимо выполнить конфигурирование нейронной сети. В рамках проведенных экспериментов

- Image size (размер изображения): параметр, определяющий длину выборки последовательности. Для различения были использованы изображения 400 на 400 пикселей. Увеличение этого параметра существенно замедляет процесс обучения при минимальном увеличении метрики Accuracy.
- Batch size (размер партии): этот параметр устанавливает количество изображений, между которыми модель нейронной сети производит поиск общих признаков за итерацию. Экспериментально оптимальным был определен параметр равный 32 изображениям. Уменьшение параметра для большинства алгоритмов (за исключением шифров KATAN64 и KTANTAN64) означало снижение точности распознавания. Значительное увеличение размера партии приводит к существенному уменьшению производительности и потенциальному переобучению модели, что также ведёт к снижению точности распознавания.

– Input shape - параметр, определяющий число нейронов на входном слое модели. В данном случае это число напрямую связано с размером изображения, так как принцип работы свёрточной нейронной сети по распознаванию изображений заключается в сложении соседних пикселей и поиске объединяющих признаков от общего к частному.

Задача по распознаванию графических отображений последовательностей напрямую используется в предлагаемом методе MLSA. В основе предлагаемого метода построения решающих правил лежит операция различения графических отображений шифртекстов друг от друга. Необходимо отметить, что для предлагаемого метода возможны несколько сценариев статистического анализа, основанных на различных по происхождению последовательностях.

Предложенные значения параметров позволили обеспечить требуемое соотношение между точностью категоризации отображений последовательностей и скоростью обучения нейронной сети.

Реализация модели СНС выполнялась на языке программирования Python и библиотек TensorFlow и FastAI.

Сценарий анализа основанный на различении соседних раундов

Сценарий анализа, основанный на сравнении соседних раундов одного и того же итеративного генератора, предполагает обучение на последовательностях каждого из раундов до R, где R - полное число раундов.

На контрольной выборке модель сравнивает шифртексты на соседних раундах и подсчитывает процент верных решений при определении принадлежности элемента контрольной выборки к той или иной группе (т.е. раунду). С увеличением числа раундов шифрования и, соответственно, улучшением статистических свойств, модель увеличивает значение E (где E – число ошибок, допущенных моделью при определении принадлежности).

Экспериментально было найдено, что линия тренда графика E с увеличением числа раундов преобразования стремится к значению 0.5. Это связано с тем, что с улучшением статистических свойств блочного шифра модель нейронной сети совершает большее число ошибок при попытке отличить элементы выборок соседних раундов шифрования. На рисунке 15 приведён график зависимости процента верных решений, принятых моделью нейронной сети при попытке различения соседних раундов от их числа на примере блочного шифра Simon.

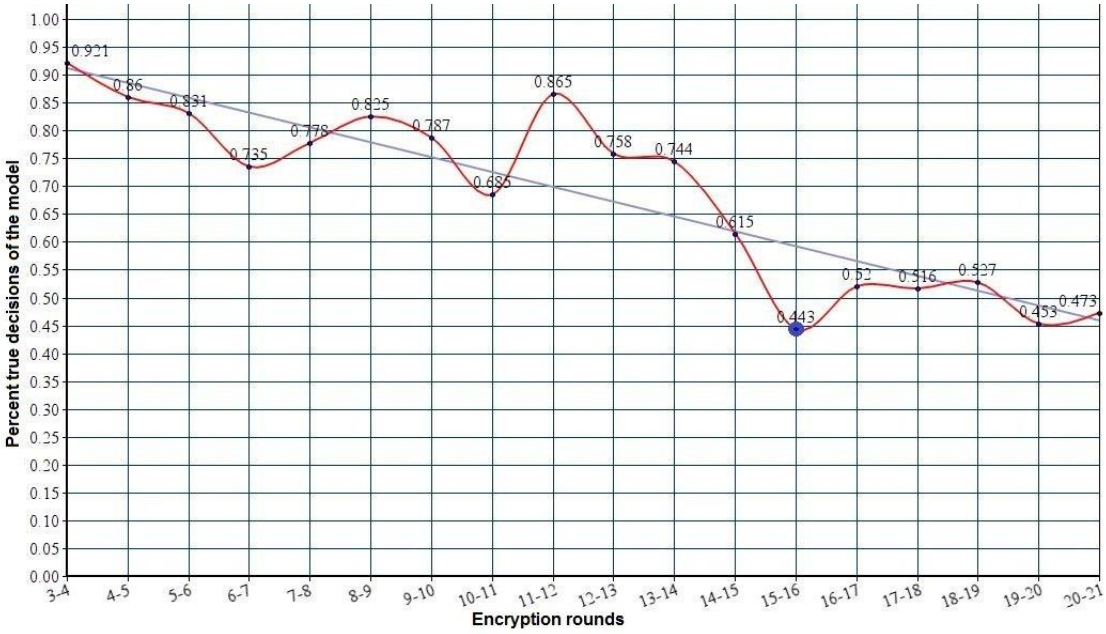


Рисунок 15– график зависимости верных решений нейронной сети от числа раундов шифрования

На графике рис. 15 видно, что на ранних раундах модель с высокой степенью вероятности различает соседние выборки. Для алгоритма Simon было найдено, что на 15 раунде (для тестов «стопка книг» и комплекса тестов NIST) достигается значение R_{min}.

Значение R_{min}, найденное классическими тестами, коррелирует с моментом, когда процент верных решений модели нейронной сети становится приближён к 0.5. Такое поведение модели объясняется тем, что на большом числе раундов последовательности становятся равномерно

распределенными и слабо отличаются друг от друга, заставляя модель работать как подбрасывание монеты.

Отличительной особенностью предлагаемого метода построения решающих правил является то, что для классических статистических тестов выборки соседних раундов слабо отличаются друг от друга: на рисунке 16 представлен график зависимости пройденных тестов NIST от числа раундов преобразования. Из графика видно, что рассматриваемый алгоритм Simon как на 3, так и на 4 раунде проходит одинаковое число статистических тестов, тогда как модель нейронной сети в 92 процентах случаев на контрольной выборке отличает шифртексты третьего раунда от шифртекстов четвертого.

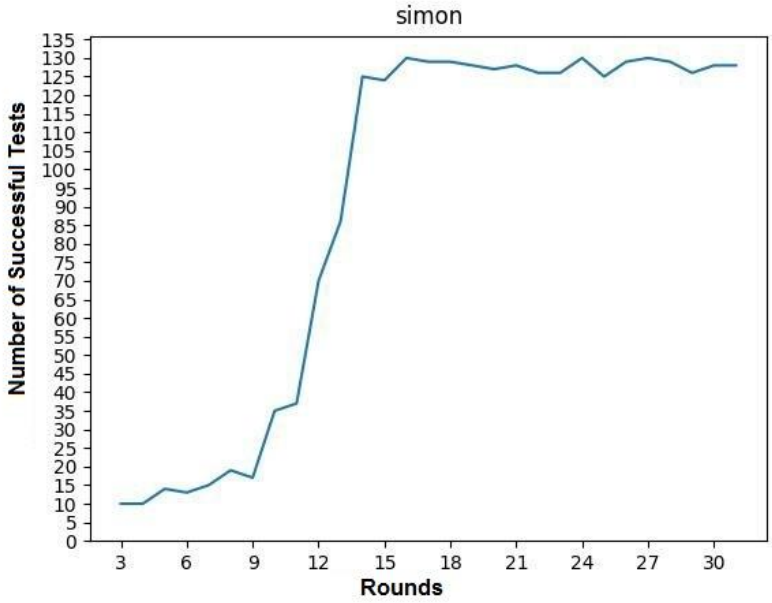


Рисунок 16 – график зависимости количества пройденных статистических тестов от числа раундов шифрования

Таким образом, можно сделать вывод о том, что метод MLSA, работающий по сценарию различения соседних раундов, может несколько лучше находить отличия в сгенерированных последовательностях, чем группа классических статистических тестов.

В таблице 3 приведены значения процента верных решений нейронной сети при различении соседних раундов итеративных блочных шифров.

Жирным выделены значения раундов, на которых значение доли верных решений попадает в доверительный интервал и соответствуют числу R_{min}.

Таблица 3 - значения правильных решений СНС для сценария соседних раундов
В строках указаны исследуемые итеративные алгоритмы. В столбцах указаны номера *раундов преобразования, которые пыталась отличить друг от друга СНС по сценарию различения соседних раундов. Жирным и подчеркнутым шрифтом выделены значения при которых достигается значение R_{min}.*

Раунд / Шифр	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18
AES	**53**	49	51	50	50	52	47	51	50	-	-	-	-	-	-
CLEFIA	61	59	**49**	51	50	53	49	51	50	46	52	53	49	51	50
DESXL	60	**46**	49	46	49	46	-	-	-	-	-	-	-	-	-
HIGHT	87	80	64	73	67	64	51	49	51	50	53	49	46	53	50
IDEA	**54**	51	47	49	51	50	46	-	-	-	-	-	-	-	-
KLEIN	**51**	53	49	51	50	46	49	51	50	46	51	55	46	49	51
LED	100	**54**	50	50	46	53	50	46	49	51	50	46	49	51	49
mCrypton	77	58	**46**	51	53	49	51	50	49	-	-	-	-	-	-
MIBS	**53**	49	51	50	46	53	50	46	49	51	50	46	51	54	50
Noekeon	**49**	46	49	46	51	53	49	51	50	53	49	51	50	-	-
Piccolo	93	60	**59**	56	47	51	50	46	51	55	46	50	49	51	50
Present	87	96	86	74	**50**	46	49	46	51	53	49	51	50	46	53
Sea	88	93	86	82	74	62	**52**	46	51	53	49	51	50	46	53
Simon	92	86	83	78	82	78	68	86	75	74	61	**44**	52	51	51
Speck	97	94	95	86	78	**51**	46	53	50	46	49	51	53	48	52
Twine	92	88	91	95	70	63	**47**	51	53	49	51	53	48	52	51
XTEA	**46**	51	53	49	51	46	49	51	50	46	51	55	46	49	51
LBlock	95	90	88	80	71	60	**49**	46	53	50	46	49	51	50	46

Из алгоритма 1 следует, что если N = 500 и α= 0.01, то $\delta = 0.06$, таким образом, при попадании значения S'^{r}_{n} в интервал [0.44;0.56] будем считать значение r = R_{min}.

Эталонный сценарий метода MLSA

Эталонный сценарий предполагает обучение свёрточной нейронной сети на сгенерированных последовательностях каждого из раундов до R, где R – полное число раундов аналогично прошлому методу и последовательностях алгоритма на R (альтернативно: на последовательностях блочного алгоритма AES 12-14 раундов).

На контрольной выборке последовательности исследуемого итеративного генератора сравниваются с выбранным эталоном. По аналогии со сценарием соседних раундов, с увеличением числа раундов генерации, число ошибок СНС увеличивается и после достижения R_{min} графические эквиваленты последовательностей становятся неотличимы от предложенного эталона. На рисунке 17 представлены графики зависимости «сценария соседних раундов» и «эталонного сценария» от числа раундов преобразования на примере итеративного блочного шифра Simon.

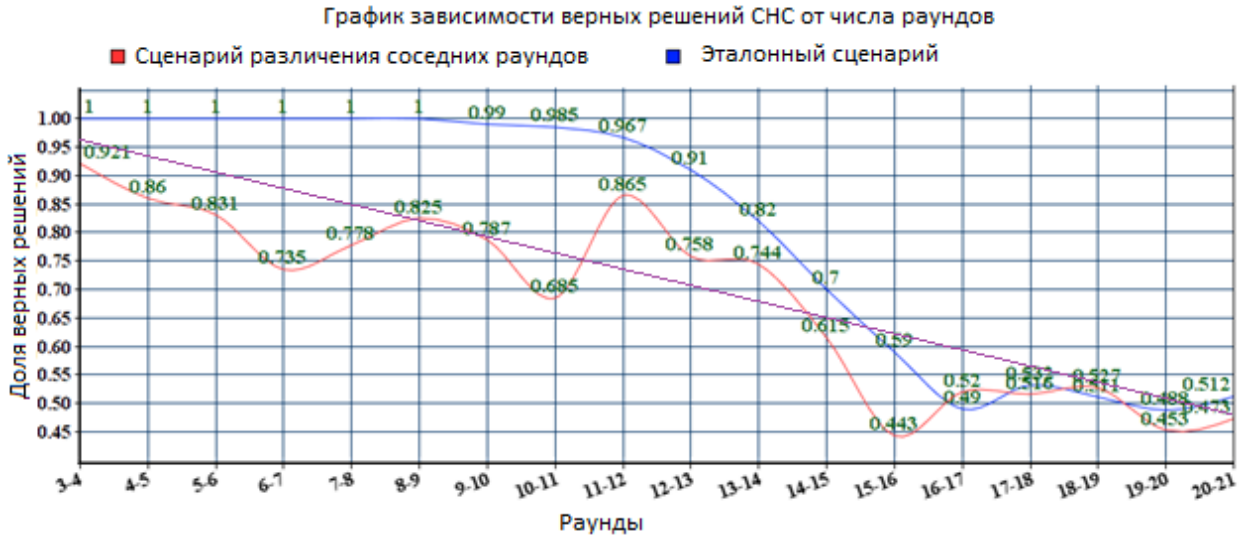


Рисунок 17 - графики зависимости верных решений СНС от числа раундов

Из графика видно, что «эталонный сценарий», в отличие от «сценария соседних раундов», на ранних раундах преобразования обладает коэффициентом верных решений СНС близким к 1. Это связано с тем, что паттерн отображений последовательностей на ранних раундах категорично отличается от отображения эталонной последовательности.

Таблица 4 - значения правильных решений СНС для «эталонного сценария»
В строках указаны исследуемые итеративные алгоритмы. В столбцах указаны номера раундов преобразования, которые пыталась отличить друг от друга СНС по «эталонному сценарию». Жирным и подчеркнутым шрифтом выделены значения при которых достигается значение R_{min}.

Раунд / Шифр	3-4	4-5	5-6	6-7	7-8	8-9	9-10	10-11	11-12	12-13	13-14	14-15	15-16	16-17	17-18
AES	**49**	46	49	46	51	53	49	51	50	-	-	-	-	-	-
CLEFIA	82	60	**51**	46	52	53	49	51	50	46	52	53	49	51	50
DESXL	59	**48**	51	46	53	50	-	-	-	-	-	-	-	-	-
HIGHT	94	87	68	73	67	64	51	49	51	50	53	49	46	53	50
IDEA	**54**	51	47	49	51	50	46	-	-	-	-	-	-	-	-
KLEIN	**51**	53	49	51	50	46	49	51	50	46	51	55	46	49	51
LED	100	**54**	50	50	46	53	50	46	49	51	50	46	49	51	49
mCrypton	92	60	**49**	51	53	49	51	50	49	-	-	-	-	-	-
MIBS	**53**	49	51	50	46	49	51	50	46	51	49	51	50	46	49
Noekeon	**52**	49	51	50	53	49	46	53	49	51	50	53	49	-	-
Piccolo	99	62	**59**	46	52	53	49	51	50	46	52	53	49	46	52
Present	100	100	87	72	**50**	49	46	51	53	49	51	50	53	49	51
Sea	100	100	96	93	74	60	**52**	47	51	50	46	51	55	46	50
Simon	100	100	100	100	100	99	98	97	91	82	70	**59**	52	53	52
Speck	100	99	99	89	72	**50**	47	51	50	46	51	55	46	50	47
Twine	100	94	97	95	77	62	**48**	46	52	53	49	51	50	46	52
XTEA	**49**	46	49	46	51	53	49	51	50	53	49	51	46	50	46
LBlock	100	99	95	91	74	58	**52**	53	50	46	49	51	53	48	53

Аналогично со «сценарием соседних раундов» определяем границу минимального числа раундов шифрования R_{min}.

Алгоритм экспериментального обоснования на примере эталонного сценария метода MLSA представлен ниже (Алгоритм 4).

Алгоритм 4. Схема экспериментального обоснования эффективности MLSA на примере «эталонного сценария»

Шаг 1: **Функция** ВычислитьОшибку (*Cipher*,*r*)
Шаг 2: Выбрать размер обучающей выборки *M*
Шаг 3: НейроннаяСеть^{r}(image) = ОбучитьНейроннуюСеть (*Cipher, r, M*)
Шаг 4: Выбрать количество контрольных выборок *N*
Шаг 5: Сгенерировать *N* контрольных выборок с помощью шифра *AES14*

и получить $\tilde{x}^{rand} = (\tilde{x}_1^{rand}, \dots, \tilde{x}_N^{rand})$

Шаг 6: Сгенерировать *N* контрольных выборок с помощью шифра Cipher и получить $\tilde{x}^r = (\tilde{x}_1^r, \dots, \tilde{x}_N^r)$

Шаг 7: Преобразовать сгенерированные множества выборок $\tilde{x}^{rand}$ **и** $\tilde{x}^r$ в изображения. Полученные множества обозначим соответственно $x^{rand} = (x_1^{rand}, \dots, x_M^{rand})$ и $x^r = (x_1^r, \dots, x_M^r)$.

Шаг 8: Экспериментально определить ошибку первого рода $E_0 = \#\{x_i^{rand} |$ Нейронная сеть$^r(x_i^{rand} = 1\}$ Экспериментально определить ошибку второго рода $E_1 = \#\{x_i^{rand} |$ Нейронная сеть$^r(x_i^{rand} = 1\}$

Шаг 9: Вернуть E_0, E_1

В таблице 5 приведены значения минимального числа раундов генерации R_{min}.

Таблица 5 - Результаты анализа алгоритмом MLSAA

Алгоритм	Full rounds	R_{min} MLSAA	$\%Rmin_R$
AES	10-14	3	21-30
CLEFIA	18-26	6	23-33
DESXL	8	4	50
HIGHT	32	10	31
IDEA	8.5	3	35
KLEIN	12-20	3	15-25
LED	32-48	4	9-12
Mcrypton	12	6	50
MIBS	32	3	9
NOEKEON	16	3	19
Piccolo	25-31	6	19-24
PRESENT	31	10	31
SEA	51	10	19
Simon	32-72	16	22-50
Speck	22-34	9	27-40
Twine	36	10	28
XTEA	32	3	9
Skipjack	12	4	30
Lblock	32	9	25

3.3 Проверка достоверности полученных результатов с помощью алгоритма Grad CAM

В целях подтверждения корректности результатов тестирования по предлагаемым методам, необходимо определить признаки, по которым модель нейронной сети различает графические представления анализируемых генерируемых последовательностей. Данный параграф посвящен верификации проведённых в предыдущих разделах экспериментов по различению графических отображений генератора.

В ряде работ, посвященных распознаванию графических изображений [83-86], уже проводились исследования, связанные с определением ключевых позиций в распознаваемом изображении путём выделения, так называемых, «важных» пикселей, изменение интенсивности которых оказывает наибольшее влияние на оценку прогноза, однако описанные в этих публикациях методы не позволяли однозначно определять класс объекта по приоритетному признаку.

Для подтверждения корректности распознавания результатов работы генератора в настоящей работе предлагается использование метода градиентного сопоставления классов (Gradient-weighted Class Activation Mapping, или GRAD-CAM метод). Выбранный подход использует информацию изображения, поступающую в конечный свёрточный слой нейронной сети для получения карты локализации важных областей в изображении. Классически такой подход используется при решении основных задач машинного зрения: с помощью Grad CAM можно выделить на изображении те признаки, по которым модель нейронной сети классифицирует изображения слонов среди множества изображений различных животных, выделяя хобот. Алгоритм создаёт тепловую карту, на которой отражается общие приоритетные признаки

Использование Grad CAM для верификации позволит однозначно определить, что алгоритм в процессе различения отображений шифртекстов

не определяет в качестве общих признаков, подаваемых на вход объектов служебную информацию файла растрового изображения (или, например, сектора изображения в начале файла).

На рисунке 18 приведён пример работы Grad CAM с изображением кошки и собаки. Алгоритм создаёт тепловую карту, на которой выделяет признаки кошки.

Рисунок 18 – пример применения алгоритма GradCAM на изображениях животных

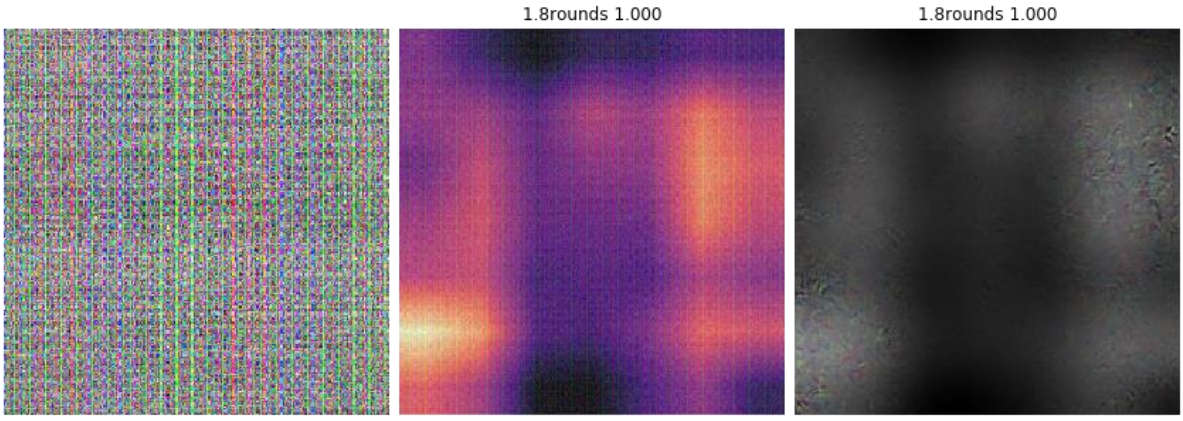


Рисунок 19 – применение алгоритма GradCAM на шифртексте алгоритма SEA

Аналогичное определение общих признаков подаваемой на вход модели последовательности сложно интерпретировать также однозначно, как изображение с предметами или животными, однако, можно сделать вывод о том, что модель не действует случайным образом при работе с генерируемыми последовательностями. На рисунке 19 приведён пример работы алгоритма Grad CAM на подаваемых на вход изображениях.

Алгоритм определил общие признаки 8 раунда алгоритма Simon и создал тепловую карту.

3.4 Система «УНИБЛОКС-2015»

Система «УНИБЛОКС-2015» [46] создана для изучения свойств выходных последовательностей итеративных блочных шифров с целью поиска параметров работы, позволяющих решать задачи криптографии успешнее, чем при параметрах, рекомендованных разработчиком.

Для достижения этой цели необходимо было обеспечить:

- подготовку базы алгоритмов для тестирования, в том числе необходимая модификация программных кодов шифров;
- объединение алгоритмов в единую структуру для систематизации процесса тестирования;
- интеграцию в информационно-аналитическую систему алгоритмов тестирования шифров;
- создание пользовательского интерфейса;
- анализ и синтез результатов тестирования, систематизация выводов.

При создании и развитии системы должно быть обеспечено выполнение основных и специальных требований, а также требований к стандартизации и унификации.

Основными требованиями, предъявляемыми к системе, являются следующие:

- наличие актуальных и современных криптографических систем защиты информации в качестве исследуемых объектов;
- корректность используемых программных реализаций шифров и хэш-функций;
- стандартизация (унификация), состоящая в едином интерфейсе используемых криптографических программных решений;
- понятность и простота оболочки (пользовательского интерфейса);
- открытость структуры, состоящая в способности системы к расширению и увеличению числа исследуемых объектов;

– достоверность исследования, состоящая в возможности верифицировать полученные результаты (провести апробацию в реальных условиях).

Начальный этап заключается в сборе исследуемых алгоритмов. На данном этапе на первый план выходит решение проблемы унификации шифров. Основу будущей библиотеки составляют легковесные шифры (lightweight ciphers), которые отличаются большим числом раундов, на каждом из которых выполняются простые преобразования. Сокращение числа раундов, при условии сохранения удовлетворительных статистических свойств, позволит ускорить работу алгоритмов, что является актуальным в условиях высокоскоростной передачи данных. В состав библиотеки вошли следующие итеративные блочные алгоритмы:

– LBLOCK [47];
– Present [48];
– XTEA [49];
– Twine [50];
– Speck [51];
– Simon [51];
– CLEFIA [52];
– Hight [53];
– Piccolo [54];
– Klein [55];
– Skipjack [56];
– mCrypton [57];
– LED [58];
– Noekeon [59];
– SEA [60];
– MIBS [61];
– IDEA [62];

– AES [63];

– DESXL [64];

– KATAN64 [65];

– KTANTAN64 [65].

Помимо вышеописанных алгоритмов к системе были добавлены недостающие шифры из библиотеки CPPCRYPTO:

– Anubis [66];

– Aria [67];

– Кузнечик [68];

– Mars [69];

– SM4 [70];

– Threefish [71].

Выбранные криптографические алгоритмы в приведённой к ним документации от разработчика имеют раздел тестовых данных (Test vectors), в котором содержатся значения открытого текста, ключа и получающегося в результате работы алгоритмы шифртекста. Данные значения приведены для проверки правильности реализации алгоритма. Реализации алгоритмов проверены данными тестовыми значениями.

Общая схема предлагаемой системы представлена на рисунке 20.

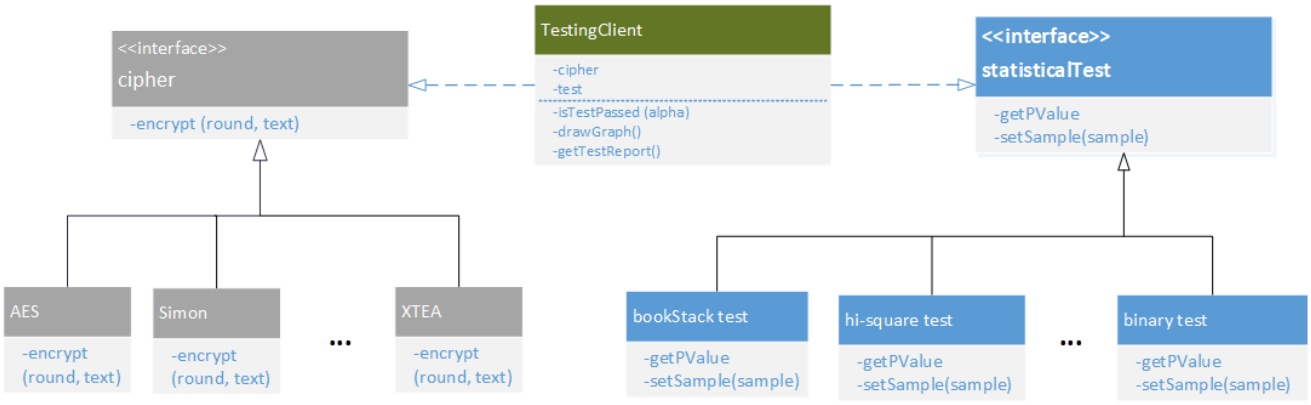


Рисунок 20 - схема системы УНИБЛОКС-2015

Собранные исходные коды алгоритмов приводятся к унифицированному виду, который позволял бы функциям формирования ключа, шифрования принимать на вход и возвращать одинаковое количество

значений единого формата. Для этого к собранным оригинальным заголовочным файлам алгоритмов добавлены библиотеки, которые бы в зависимости от оригинальной реализации шифра приводили бы его к единому виду при помощи «обёрточных функций». Для этого каждая оригинальная реализация шифра получила к своему названию префикс «_orig», которая вызывалась в основном заголовочном файле шифра, а преобразующие заголовочные файлы получали имена в соответствии с шифром (например, файлы lblock.h и lblock_orig.h). Основные файлы шифров содержат функции формирования ключей, а функции шифрования используют функции преобразования ключей к единому виду. Ниже приведён фрагмент кода заголовочного файла led.h, реализующего шифр LED.

```
void encrypt(u32* text, int rounds) {
     _32to8(input, text, 2);//преобразование слов
     LED_enc(input, rounds);
     _8to32(input, text, 2);//обратное преобразование
}
```

Функция encrypt шифра, которая находится в заголовочном файле led_orig.h работает с шестнадцатью 8-битовыми словами, а библиотека работает с четырьмя 32-битовыми. Для конвертации входных данных используется библиотека utils.h, которая содержит алгоритмы преобразования всех видов входных данных в шифрах. В приведённом выше фрагменте кода показано, как прежде чем вызвать функцию шифрования, происходит вызов функции преобразования 32-битных слов в 8-битные. Вызывается шифрование, и, затем, происходит обратное преобразование к виду, с которым уже работает библиотека.

Отдельно необходимо отметить то, как происходит преобразование функций к единому виду. Написанные разными разработчиками шифры имеют разные сигнатуры. Например, на вход функции шифрования принимается разное количество аргументов (шифруемый текст, количество раундов, ключ, длина ключа и др.). Для правильного функционирования

библиотеки ставится задача унификации всех алгоритмов к единому виду. Для этого используются «обёрточные» функции, которые возможно реализовать с помощью шаблона проектирования «адаптер». *Адаптер* [72] - структурный шаблон проектирования, предназначенный для организации использования функций объекта, недоступного для модификации через специально созданный интерфейс. Применяется в случае, если функция (класс) поддерживает требуемые данные и поведение, но имеет неподходящий интерфейс. Для использования создается новая оболочка функции с требуемым интерфейсом. Ниже приведён код шифра AES (файл aes.h), в котором используется «обёрточная» функция.

```
void encrypt(unsigned int* text, int rounds) {
     _32to8(in, text, 4); //преобразование входных слов
     Cipher(rounds);//вызов оригинальной функции
     _8to32(out, text, 4);// преобразование выходных слов
}
```

Функция Cipher, которая содержится в оригинальной реализации шифра AES (файл aes_orig.h), принимает на вход одно значение – количество раундов шифрования. Библиотека вызывает у каждого шифра функцию шифрования с двумя параметрами – открытым текстом и числом раундов шифрования. Для того, чтобы привести все функции к единому виду, вызывается функция encrypt с необходимыми для работы библиотеки параметрами, внутри которой происходит вызов оригинальной функции шифрования с параметрами, с которыми работает данная функция, а также все преобразования входного текста к требуемой длине слов. В данном примере преобразование идёт от «меньшего» к «большему» (от одного аргумента к двум). Ниже приведён фрагмент кода алгоритма CLEFIA, который выполняет преобразование, от «большего» к меньшему (приводит функцию с четырьмя аргументами к функции с двумя аргументами).

```
void encrypt(unsigned int* text, int rounds) {
     _32to8(text8, text, 4); //преобразование входных слов
     ClefiaEncrypt(text8, text8, rk, rounds);
```

```
    _8to32(text8, text, 4); // преобразование выходных слов
}
```

Решая проблему унификации, приходится сталкиваться с проблемой одинаковых имён (во многих шифрах функции имеют одинаковые имена, например – encrypt), а так как все заголовочные файлы шифров подключаются, в конечном итоге, к одному файлу библиотеки, происходит конфликт имён, вследствие чего компилятор фиксирует ошибки в написании программного кода. Для решения этой проблемы используется другой шаблон проектирования – стратегия. *«Стратегия»* [73] - поведенческий шаблон проектирования, предназначенный для определения семейства алгоритмов, инкапсуляции каждого из них и обеспечения взаимозаменяемости. «Стратегия» применяется в случаях, когда в одном и том же месте программы, в зависимости от текущего состояния, необходимо использовать различные функции или алгоритмы. Ниже приведены фрагменты кода шифров Klein и IDEA, которые вызывают разные шифры при одинаковых названиях функций шифрования.

```
void encrypt(unsigned int* text, int rounds) {//шифр Klein
    _32to8(text8, text, 2);
    Encrypt(text8, crypt, k8, rounds);
    _8to32(crypt, text, 2);
}
void encrypt(u32* text, int rounds) {// шифр IDEA
    _32to16(state, text, 2);
    Encrypt(state, subkey, rounds);
    _16to32(state, text, 2);
}
```

Как видно из этого фрагмента кода, функции имеют одинаковое название, следовательно, при компилировании появляется проблема повторного объявления одной и той же функции. Чтобы избежать подобной ошибки, необходимо использование пространства имён, которое позволит в заголовочном файле библиотеки вызывать функции с одинаковыми именами в разных ситуациях, в зависимости от выбранного при тестировании шифра.

Для реализации «стратегии» каждому алгоритму в своём заголовочном файле зададим собственное пространство имён:

```
namespace Idea {
     ... кодфайла idea.h
}
namespace Klein {
     ... кодфайла klein.h
}
```

Такое объявление позволит вызывать функции с помощью указания имени после определения пространства имён, в котором эта функция находится. Когда в файле библиотеки будет указываться идентификатор шифра, программа будет использовать библиотеку blockciphers.h, в которой указаны необходимые функции, в соответствии с пространствами имён и присвоенными им идентификаторами. Ниже приведены фрагменты кода основного заголовочного файла библиотеки Ciphers.h и вспомогательной библиотеки blockciphers.h.

```
#include "blockciphers.h" // подключение blockciphers.h
int main(int argc, char **argv){
          ...
          setCipher(Ciphers::KLEIN);//выбора алгоритма
          ...
}
void setCipher(Ciphers cipher) {
     ...
     else if (cipher == KLEIN) { //условие выбора шифра KLEIN
               encrypt = Klein::encrypt; //
               resetKey = Klein::resetKey;
               id = Klein::id;
               runTests = Klein::runTests;
     }
     else if (cipher == IDEA) { // условие выбора шифра IDEA
          encrypt = Idea::encrypt; //
          resetKey = Idea::resetKey;
```

```
            id = Idea::id;
            runTests = Idea::runTests;
      }
}
```

Таким образом, обращение к конкретной функции шифрования происходит явно, что позволяет избежать множественного объявления функции.

Помимо пространства имён библиотека blockciphers.h используют указатели. Указатели предназначены для хранения адресов областей памяти, а ссылка на них представляет собой синоним имени, указанного при инициализации ссылки. Для создания тестирующей библиотеки необходимо наличие возможности циклического запуска одновременно нескольких шифров для сравнения их статистических характеристик. Для того, чтобы однажды указать идентификатор шифра и включить функцию инициализации ключа и функцию шифрования, необходимо сделать указатели на эти функции, которые объявлены в библиотеке blockciphers.h. Ниже приведён фрагмент кода, используемый для объявления указателей.

```
void(*encrypt)(u32* text, int rounds);
void(*resetKey)();
char*(*id)();
void(*runTests)();
```

При указании идентификатора шифра в функции setCipher(), в зависимости от передаваемого значения, происходит множественный выбор и вызов функций по указателю:

```
void setCipher(Ciphers cipher) {
      if (cipher == LBLOCK) {
            encrypt = Lblock::encrypt;
            resetKey = Lblock::resetKey;
            id = Lblock::id;
            runTests = Lblock::runTests;
      }
}
```

Важным этапом разработки является создание библиотеки utils.h, задачей которой является преобразование входных данных шифров. Ниже приведён фрагмент кода, который реализует разделение длинных слов на короткие и соединение коротких слов в длинные. Как уже отмечалось выше, библиотека по умолчанию работает с 32-битовыми словами.

```
void _16to32(u16* text16, u32* text32, int n_words32) {
     for (int i = 0; i < n_words32; i++) {
          text32[i] = text16[2 * i];
          text32[i] <<= 16;
          text32[i] += text16[2 * i + 1];
     }
}
void _32to16(u16* text16, u32* text32, int n_words32) {
     for (int i = 0; i < n_words32; i++) {
          text16[2 * i] = (text32[i] >> 16) & 0x0000FFFF;
          text16[2 * i + 1] = text32[i] & 0x0000FFFF;
     }
}
```

Библиотека utils.h содержит следующие виды преобразований входных данных:

- 32-битные слова склеиваются в 64-битные;
- 16-битные слова склеиваются в 32-битные;
- 8-битные слова склеиваются в 32-битные;
- 4-битные слова склеиваются в 32-битные;
- 64-битные слова разделяются на 32-битные;
- 32-битные слова разделяются на 16-битные;
- 32-битные слова разделяются на 8-битные;
- 32-битные слова разделяются на 8-битные.

Библиотеки utils.h, blockciphers.h, а также заголовочные файлы алгоритмов шифрования подключаются к файлу библиотеки Ciphers.cpp, в которой и происходит основное управление тестированиями. Код библиотеки

utils.h приведён в Приложении А.

Библиотека позволяет изменять число раундов шифрования, значения шифруемых блоков, количество шифруемых блоков. При необходимости пользователь может явным образом задать ключ в файлах того или иного шифра, как это происходило при вызове процедуры runTests. Ниже приведены фрагменты кода, изменяющие параметры тестирования.

```
int rnds = 12; // раунды
setCipher(LBLOCK); //процедура вызова шифра LBlock
```

В следующем примере первые два значения шифруемых блоков задаются с помощью генератора случайных чисел rand, а значения blk[2] и blk[3] задаются в явном виде.

```
blk[0] = rand() % 255;
blk[1] = rand() % 255;
blk[2] = 0x123456;
blk[3] = 0x456789;
```

Следующий пример демонстрирует явное присваивание значений ключа, что может потребоваться для проведения теста на корректность реализации криптографического алгоритма.

```
k8[0] = 0x12;
k8[1] = 0x34;
k8[2] = 0x56;
k8[3] = 0x78;
```

С помощью изменения параметров шифрования можно сократить число раундов шифрования для увеличения скорости работы алгоритма, что является одной из основных задач создания информационно-аналитической системы.

Подключение новых алгоритмов

Преимуществом создания библиотеки, помимо удобного тестирования алгоритмов, является возможность добавления новых алгоритмов при

соблюдении условий, описанных выше. Алгоритм должен быть приведён к требуемому виду, заголовочные файлы шифра должны быть подключены к головному файлу библиотеки с помощью директивы include. Ниже приведён пример подключения нескольких заголовочных файлов шифров.

```
#include "mcrypton.h"
#include "led.h"
#include "noekeon.h"
#include "sea.h"
#include "mibs.h"
#include "idea.h"
#include "aes.h"
#include "desxl.h"
```

Также для подключения нового шифра необходимо добавить идентификатор с помощью оператора перечислений enum. Ниже приведён фрагмент кода оператора enum.

```
enumCiphers {
     LBLOCK,
     PRESENT,
     XTEA,
     TWINE,
     ...
}
```

Последним шагом при подключении нового шифра к библиотеке является добавление кода к условному оператору, отвечающему за вызов функций. Ниже приведён фрагмент кода процедуры setCipher(), которая содержит в себе условный оператор if-else-if, предназначенный для вызова необходимых функций.

```
void setCipher(Ciphers cipher) {
     if (cipher == LBLOCK) {
          encrypt = Lblock::encrypt;
          resetKey = Lblock::resetKey;
          id = Lblock::id;
          runTests = Lblock::runTests;
```

```
    }
    else if (cipher == PRESENT) {
        encrypt = Present::encrypt;
        resetKey = Present::resetKey;
        id = Present::id;
        runTests = Present::runTests;
    }
    else if (cipher == XTEA) {
        encrypt = Xtea::encrypt;
        resetKey = Xtea::resetKey;
        id = Xtea::id;
        runTests = Xtea::runTests;
    }
    else if (cipher == TWINE) {
        encrypt = Twine::encrypt;
        resetKey = Twine::resetKey;
        id = Twine::id;
        runTests = Twine::runTests;
    }
}
```

Данная процедура использует описанные выше переменные ссылочного типа, а также пространства имён. Принимая на вход идентификатор алгоритма, процедура указывает функции, которые следует вызвать для текущего тестирования.

Таким образом, использование программной библиотеки подразумевает вызов четырех функций: setCipher, id, resetKey и encrypt. Для каждого шифра также предусмотрена вспомогательная функция runTests, где вызываются проверочные тесты по правильности реализации шифра. В этой функции, например, могут проверяться тестовые вектора из статьи, где предложен шифр.

В библиотеке существует перечисление Ciphers, задающее константы для имеющихся шифров: enum Ciphers {LBLOCK, PRESENT, …, DESXL}.

Алгоритм использования библиотеки следующий.

– вызвать функцию setCipher, передав константу, соответствующую исследуемому шифру;

– опционально можно вывести идентификатор шифра, который возвращает функция id;

– вызвать функцию resetKey, которая задает случайный мастер-ключ и выполняет процедуру развертывания ключа (создание ключей раундов);

– сформировать входной блок и вместе с числом раундов передать его функцию encrypt;

– обработать выходной блок согласно алгоритму статистического теста.

Унификация программных интерфейсов блочных шифров заключалась, главным образом, в решении следующих задач:

– приведение входных и выходных блоков, представленных в виде 32-битовых слов, к представлению, требуемому в конкретных реализациях;

– вынесение числа раундов в список аргументов функции шифрования;

– создание функциональности для выбора нового случайного ключа.

Исходный программный код библиотеки blockciphers.h приведён в Приложении Б.

3.5 Реализация и применение решающих правил тестом «стопка книг»

Одним из тестов является статистический тест для случайных чисел «стопка книг» разработанный и опубликованный в 2004 году [74].

Нулевая гипотеза H_0 традиционно предполагает, что все буквы исходного алфавита равновероятно, тогда как альтернативная является отрицанием H_0.

Тестирование "стопкой книг" предполагает, что исходный алфавит пронумерован в соответствии с порядком букв, и по ходу теста порядок меняется после анализа каждого значения, увеличивая значение номера более часто появляющихся букв, таким образом, тестирование сводится к подсчёту частоты появления номеров букв исходного алфавита, и, в абсолютно идеальной последовательности, вероятность появления в выборке буквы с любым номером равна 1/S. Результатом теста является вычисленное значение χ^2.

Известно, что распределение случайной величины асимптотически приближается к распределению χ^2 с r-1 степенью свободы при выполнении гипотезы H_0. Как правило, значения «хи-квадрат» теста интерпретируются следующим образом: значения, располагающиеся в интервале от 0 до 3.84 включительно, являются незначимыми: вероятность того, что реально наблюдаемый результат является чисто случайным, равна по крайней мере 1 из 20; значения выше 3.84 являются потенциально значимыми; уровень достоверности в этом случае — 95%; таким образом, существует менее 1 из 20 шансов за то, что появление наблюдаемых результатов было чисто случайным.

Ниже приведён фрагмент кода, реализующий подсчёт критерия χ^2 в тесте «стопка книг».

```
double Chi(int skolko){
          double np;
          int pow = s1 - n1 - k1;
          if (pow < 0) pow = -pow;
          np = skolko*Power(2, pow);//подсчёт NP
          return (nu - np)*(nu - np) / (1.0*np);
}
void CalcChi(int outs) {
          x2 = Chi(outs); sumx2 = sumx2 + x2
          if (x2 > 3.84) u05 = u05 + 1;
          if (x2 > 6.64) u01 = u01 + 1;
}
```

Выбор количества подмножеств A_j, их величины определяются экспериментально. При тестировании генераторов случайных и псевдослучайных чисел, как и в некоторых других задачах, возможно проведение специальных экспериментов для определения значений этих величин, а затем проведение проверки гипотез по независимым данным (или по участкам последовательности псевдослучайных чисел, не использовавшимся при подборе указанных значений). Такие эксперименты направлены на поиск значений параметров, позволяющих выявить отклонения от случайности при приемлемых объемах выборки (или за приемлемое время вычислений).

Ниже приведены фрагменты кода библиотеки с комментариями, описывающими основные этапы тестирования.

```
srand(time(0));
u32 blk[4];
unsigned int i;
keyquo = 10;
Init(18, 24, 32);
for (keyn = 0; keyn < keyquo; keyn++) {
     resetKey();
     nu = 0;
     for (i = 0; i < N1; i++){
          blk[0] = 0;
          blk[1] = i;
          blk[2] = 0;
          blk[3] = 0;
          encrypt(blk, rnds);
          Search(blk[0]); nu = nu + is;
     }
          CalcChi(1);
}
```

После проведенного расчёта программа выводит статистику по каждому шифру на каждом раунде: процент неудовлетворительных выборок по критерию χ^2 как по значению 3.84, так и по значению 6.64. Среднее

значение χ^2 по выборкам, а также пройден ли тест по данной выборке. Ниже приведён код процедуры getStats(), которая отвечает за получение статистики по выборке.

```
Stats getStats() {
     avx2 = sumx2 / keyquo;
     perc05 = 100.0*u05 / keyquo;
     perc01 = 100.0*u01 / keyquo;
     sumx2 = 0; u05 = 0; u01 = 0;
     return{ perc01, perc05, avx2 };
}
```

Функция вызывается в заголовочном файле теста, фрагмент вызова и вывода значений приведён ниже.

```
Stats st = getStats();//вызов функции
fprintf(rfile, "%10.0f %10.0f %10.2f", st.perc05, st.perc01,
st.avx2);
printf("%10.0f %10.0f %10.2f", st.perc05, st.perc01, st.avx2);
```

Полный код теста приведён в Приложении В. На рисунке 17 приведён пример запуска запрограммированного теста. Полный код процедур теста bookstack.h приведён в Приложении Г.

При тестировании для каждого шифра варьируется число раундов 1,2,..., R. Для каждого раунда генерируется по 10 выборок (10 случайных ключей) размера 2^{26} байт, и по каждой вычисляется статистика χ^2. Выборки генерируются следующим образом: в зависимости от шифра блоки представляются в виде 2-х,3-х или 4-х 32-битовых слов, и в качестве элементов выборки используются первые слова каждого выходного блока. На вход шифра подаются блоки, где все слова, кроме второго, равняются нулю, а второе слово блока изменяется от 0 до 2^{24}-1. Значения статистики χ^2 сравнивались с квантилью «хи-квадрат» уровня значимости 0.05 с одной степенью свободы. При этом для каждой серии из 10 таких выборок подсчитывается число $U_{0.05}$, означающее количество превышений статистикой χ^2 данной квантили.

Поскольку статистические свойства шифра улучшаются с ростом числа раундов, то при определенном числе раундов распределение выборки не будет отличаться от равномерного. Данное число (обозначим его R_{min}) берём в качестве предварительной оценки для R_{min}. Эксперименты показывают, что в среднем, рассмотренные блочные шифры сохраняют удовлетворительные статистические свойства при сокращении числа раундов в них до 10-30% от полного числа раундов. В таблицах 6 и 7 приведены результаты тестирования с помощью теста «стопка книг». Знак «+» обозначает, что результаты шифрования неслучайны на 8-10 ключах, а знак«-» обозначает, что результаты шифрования неслучайны на 0-2 ключах.

Таблица 6 - результаты работы теста«стопка книг»

Rounds	1-14	15	16	17	18-24	25	26-27	28-29	30	R_{min}	R	%
Simon	+	-	-	-	-	-	-	-	-	16	32-72	25-56
KATAN64	+	+	+	+	+	+	-	+	-	30	254	12
KTANTAN64	+	+	+	+	+	+	-	+	-	30	254	12

Таблица 7 - результат работы теста «стопка книг»

Rounds	1	2	3	4	5	6	7	8	9	10-13	14	R_{min}	R	%
XTEA	+	+	-	-	-	-	-	-	-	-	-	3	32	6
SPECK	+	+	+	+	+	-	-	-	-	-	-	6	22-34	15-23
CLEFIA	+	+	+	+	+	-	-	-	-	-	-	6	18-26	19-28
Piccolo	+	+	+	+	+	-	-	-	-	-	-	6	25-31	16-20
KLEIN	+	+	+	-	-	-	-	-	-	-	-	4	12-20	10-17
mCrypton	+	+	+	-	+	-	-	-	-	-	-	6	12	25
LED	+	+	+	-	-	-	-	-	-	-	-	4	32-48	6-10
Noekeon	+	-	-	-	-	-	-	-	-	-	-	2	16	6
MIBS	+	-	-	-	-	-	-	-	-	-	-	2	32	3
IDEA	+	-	-	-	-	-	-	-	-	-	-	2	8.5	12
AES	+	+	+	-	-	-	-	-	-	-	-	4	10-14	21-30
DESXL	+	-	-	-	-	-	-	-	-	-	-	2	8	25

LBlock	+	+	+	+	+	+	+	-	-	-	-	8	32	25
Present	+	+	+	+	+	+	+	+	-	-	-	9	31	26
Twine	+	+	+	+	+	+	+	+	-	-	-	9	36	22
Hight	+	+	+	+	+	+	+	+	+	-	-	10	32	28
Sea	+	+	+	+	+	+	+	+	+	-	-	10	51	18
Skipjack	+	+	+	+	+	+	+	+	+	+	-	14	32	34

Тестирование «стопкой книг» показало возможность существенного снижения числа раундов шифрования при условии сохранения удовлетворительных статистических свойств шифров.

3.6 Применение решающих правил на примере теста «Адаптивный критерий χ2»

Задача проверки гипотез является одной из центральных для математической статистики и теории информации и является прикладной задачей для в криптографии, где случайные числа и методы их тестирования используются очень широко. Как уже было отмечено в п.2.1, одна из задач такого типа в криптографии – эффективное тестирование последовательностей, зашифрованных некоторым блоковым шифром. Точнее, одно из требований, предъявляемых к современным блоковым шифрам, – неотличимость зашифрованной последовательности от случайной даже в том случае, когда исходные данные заведомо неслучайны. Пример задачи такого типа, представляющей и некоторый самостоятельный интерес для криптографии, - различение зашифрованных текстов на естественных языках от случайных последовательностей.

В данном разделе тестирование проводится по адаптивной схеме критерия χ^2, мощность которого при некоторых типах альтернатив существенно выше, чем у «обычного» метода χ^2. Предлагаемый критерий позволяет отличить зашифрованные тексты на русском языке от случайных при меньшем объеме выборки, чем обычный метод χ^2.

Ниже приведены фрагменты кода реализации теста «адаптивный критерий χ^2». На первом этапе происходит формирование обучающей выборки: блоки с одним ненулевым словом шифруются и поочередно заносятся в хэш-таблицу, которая хранит в себе значения.

```
setCipher(Ciphers::KTANTAN64); //выбор тестируемого шифра
hash_set <unsigned long > hs1;//объявление хэш-таблиц
hash_set <unsigned long > ::const_iterator hs1_AcIter,
hs1_RcIter;
int k = 0
resetKey();
for (int i = 0; i < 1048576; i++) {
          blk[0] = 0;
          blk[1] = i;//ненулевое слово блока
          blk[2] = 0;
          blk[3] = 0;
}
encrypt(blk, rnds);
hs1.insert(blk[1]);
```

На втором этапе происходит шифрование блоков от 2^{20} до $2^{24}+2^{20}$, происходит формирование контрольной выборки. Зашифрованное слово ищется в «обучающей» выборке, и в случае нахождения соответствия значение переменной k увеличивается на 1. Эта переменная хранит в себе число встреч a_i в выборке $x_1,\ldots,x_n$. Ниже приведён фрагмент программного кода теста, формирующий «контрольную» выборку.

```
for (inti = 1048576; i< 17825792; i++) {
               blk[0] = 0;
               blk[1] = i;
               blk[2] = 0;
               blk[3] = 0;
}
encrypt(blk, rnds);//вызов функции шифрования
hs1_RcIter = hs1.find(blk[1]);
if (hs1_RcIter == hs1.end){
```

```
        k += 0;
        else
        k++;//иначе прибавляем +1
}
cout << k << " ";
hs1.clear();
```

В результате, тест выводит значения k_i, где i – порядковый номер эксперимента, каждый из которых проходит на новом, генерируемым случайным образом ключе шифрования. Как и в критерии «стопка книг», в случае, если на более чем двух ключах тест показывает недопустимое количество отклонений от равномерного распределения, считается что шифр не прошёл тест на данном показателе числа R. На рисунке 5 приведен пример проведения теста, а в таблице 5 сведены результаты выполнения теста для шифра AES. Код теста приведён в Приложении Д.

Результаты тестирования шифра AES показали, что на 3-м раунде шифр показывает удовлетворительные статистические свойства на 8 из 10 ключах. Уже на четвертом раунде статистика шифра улучшается до 10 ключей, что является аналогичным результатом с тестом «стопка книг». В таблицах 8 и 9 указаны сводные данные всех шифров, над которыми проводились эксперименты с помощью теста «Адаптивный критерий χ^2».

Таблица 8 - результаты теста «адаптивный критерий хи-квадрат»

Rounds	1-14	15	16	17	18-24	25	26-27	28-29	30-32	R_{min}	R	%
KATAN64	+	+	+	+	+	+	+	+	-	31	254	12
KTANTAN64	+	+	+	+	+	+	+	+	-	31	254	12

Таблица 9 - результаты теста «адаптивный критерий хи-квадрат»

Rounds	1	2	3	4	5	6	7	8	9	10-13	14	R_{min}	R	%
XTEA	+	+	-	-	-	-	-	-	-	-	-	3	32	6
SPECK	+	+	+	+	+	-	-	-	-	-	-	6	22-34	15-23
CLEFIA	+	+	+	+	+	-	-	-	-	-	-	6	18-26	19-28

Piccolo	+	+	+	+	-	-	-	-	-	-	-	5	25-31	16-20
KLEIN	+	+	+	-	-	-	-	-	-	-	-	4	12-20	10-17
mCrypton	+	+	+	-	-	-	-	-	-	-	-	3	12	25
LED	+	+	+	-	-	-	-	-	-	-	-	4	32-48	6-10
Noekeon	+	-	-	-	-	-	-	-	-	-	-	2	16	6
MIBS	-	-	-	-	-	-	-	-	-	-	-	1	32	3
IDEA	+	-	-	-	-	-	-	-	-	-	-	2	8.5	12
AES	+	+	+	-	-	-	-	-	-	-	-	4	10-14	21-30
DESXL	+	+	+	-	-	-	-	-	-	-	-	4	8	25
LBlock	+	+	+	+	+	+	-	-	-	-	-	7	32	25
Present	+	+	+	+	+	+	+	-	-	-	-	8	31	26
Twine	+	+	+	+	+	+	+	+	-	-	-	9	36	22
Hight	+	+	+	+	+	+	+	+	+	-	-	10	32	28
Sea	+	+	+	+	+	+	+	+	+	-	-	10	51	18
Skipjack	+	+	+	+	+	+	+	+	+	+	-	14	32	34
Simon	+	+	+	+	+	+	+	+	+	-	-	12	32-72	25-56

Результаты испытаний нескольких шифров отличаются от результатов экспериментов с помощью теста «стопка книг». Тест «стопка книг» разработан несколько позже и, по мнению разработчиков, является более точным и строгим тестом, чем «Адаптивный критерий χ^2». В общем, показатели R_{min} двух статистических тестов коррелируют между собой. Расхождения в результатах происходят, в основном, не более чем на один раунд, за исключением шифра SIMON, минимальное число раундов которого на двух разных тестах отличается на 6.

3.7 Применение решающих правил на примере тестов NIST

Ряд статистических тестов разработан NIST [45]. В основе каждого из этих тестов лежит задача вычисления статистики, характеризующей некое свойство последовательности, после чего эта статистика сравнивается с эталонной статистикой, которую даёт идеально случайная

последовательность. Эталонная статистика выводится математически, чему посвящено множество теорем и научных трудов по криптографии и теории чисел. Тесты NIST в рамках исследований выходных последовательностей криптографических систем применялись в работе [76], рекомендации по их применению указаны в [77-79].

В основе тестов лежит понятие нулевой гипотезы. Нулевая гипотеза - это предположение, что между фактами появления чисел имеется какая-либо взаимосвязь. Иными словами, за нулевую гипотезу принимается предположение, что последовательность является истинно случайной (знаки которой появляются равновероятно и независимо друг от друга). Следовательно, если такая гипотеза верна, то генератор отвечает хорошим статистическим характеристикам.

Проверка гипотезы заключается в том, что имеется статистика, подсчитанная на основе собранных данных (например, по уже сгенерированным ключам). С другой стороны, имеется эталонная статистика, получаемая математическими методами (вычисленная сугубо теоретически), которая бы имела идеально случайная последовательность. Фактическая статистика, конечно, не может сравняться с эталонной: насколько бы ни был хорош генератор, он не может быть идеален. Для этого вводится некоторая доля погрешности (например, 0.05). Таким образом, существует 4 итоговых результата:

- Сделан вывод о том, что последовательность случайна, и это верный вывод;
- Сделан вывод о том, что последовательность не случайна, хотя она была на самом деле случайна. Такие ошибки называют ошибками первого рода;
- Последовательность признана случайной, хотя на самом деле таковой не является, такие ошибки называют ошибками второго рода;
- Последовательность справедливо отбракована.

В каждом тесте вычисляется так называемое P-значение: это

вероятность того, что генератор произведет последовательность не хуже, чем гипотетический истинный. Если P-значение = 1, то последовательность идеально случайна, а если оно = 0, то последовательность полностью предсказуема. В дальнейшем P-значение сравнивается с α, и если она больше α, то нулевая гипотеза принимается и последовательность признается случайной. В противном случае — отбраковывается.

В тестах берется α = 0.01. Из этого следует, что:

- Если P-значение ≥ 0.01, то последовательность признается случайной с уровнем доверия 99%;
- Если P-значение < 0.01, то последовательность отбраковывается с уровнем доверия 99%.

Для проведения тестов в систему были дополнительно интегрированы шифры, представленные в библиотеке CPPCRYPTO, а также 15 статистических тестов NIST.

В рамках существующей реализации тесты NIST работают на базе библиотеки GSL (GNU Scientific Library). GSL - это библиотека написанная на языке C, которая содержит значительное количество функций - от элементарных математических операций и операций с комплексными числами до численных методов дифференцирования, интерполяции, аппроксимации, решения дифференциальных уравнений, wavelet-преобразования и многих других.

Технология взаимодействия статистических тестов и алгоритмов

Одной из ключевых задач разработанной системы является обеспечение взаимодействия статистических тестов и криптографических алгоритмов. Для выполнения этой задачи был разработан скрипт, написанный на языке Python, который автоматизировал процесс тестирования за счёт рекурсивного обхода каталогов с файлами сформированных шифртекстов и запускал процесс анализа шифров. После

прохождения тестов происходит автоматическое формирование отчётов о тестировании, включающих информацию о результатах, тестовые значения и результирующие параметры. Ниже приводится фрагмент программного кода скрипта. С полной версией программного кода можно ознакомиться в приложении Г.

```
     For filepath in result: reports_folder =
filepath.parent.joinpath('reports')
               if not reports_folder.exists():
                    reports_folder.mkdir()

               input_file = str(filepath)
               output_file =
reports_folder.joinpath(filepath.name.replace('.bin', '.txt'))

               args = ("./statisticaltests", input_file,
output_file)  # запускаем тесты, указываем файл с
   p = subprocess.Popen(args, stdout=subprocess.PIPE)
         p.wait()
```

После прохождения тестов генерируется выходной файл с отчётом, отличающимся от исходного шифртекста расширением .txt. В файл отчёта записываются полные данные о прошедшем тесте. Например, для шифра ktantan64 с тремя раундами фрагмент отчёта выглядит следующим образом:

```
Frequency test   FAILURE
Nb. of ones: 636280
Nb. of zeros: 363720
P-value: 0
Block frequency test  FAILURE
Nb. of blocks: 50
Block length: 20000
Nb. of discarded bits: 0
P-value: 0
```

В данный отчёт собираются все значения проведённых над шифртекстом тестов. Для однозначной трактовки результатов теста также формируются также короткие отчёты с помощью вышеописанного скрипта, фрагмент которого приводится ниже.

```
result_types = ('SUCCESS', 'FAIL')
```

```
   short_report_file =
reports_folder.joinpath(filepath.name.replace('.bin',
'_short.txt'))
          with output_file.open('r') as report,
short_report_file.open('w') as short_report:
                    for line in report.readlines():
                         line = str(line)
               if any(substr in line for substr in result_types):
                    short_report.write(line)
```

Описание тестов NIST

Ниже приводится описание интегрированных в систему тестов NIST. В каждом из тестов вычисляется значение P (P-value). Если вычисленное в ходе теста значение вероятности $p < 0{,}01$, то данная двоичная последовательность не является абсолютно случайной. В противном случае, она носит случайный характер.

Частотный побитовый тест (Frequency (Monobit) Test)

В основе теста лежит идея поиска соотношения нулей и единиц в исследуемой последовательности. Тест оценивает близость доли единиц к 0.5. Следует отметить, что все последующие тесты так или иначе зависят от результата частотного побитового теста и проводятся только в случае успешного прохождения.

Частотный блочный тест (Frequency Test within a Block)

Целью теста является определение доли единиц внутри блока длиной m бит. Тест определяет, насколько близко число единиц в блоке к m/2, как предполагается при абсолютно случайной последовательности.

Тест на последовательность одинаковых битов (Runs test)

Производится подсчёт полного числа рядов в исследуемой последовательности, где под словом "ряд" подразумеватся непрерывная последовательность одинаковых бит. Ряд длиной k состоит из k одинаковых бит, начинается и заканчивается с противоположного бита. Цель теста - сделать вывод о том, действительно ли количество рядов, состоящих из

нулей и единиц с различными длинами, соответствует их количеству в случайной последовательности.

Тест на самую длинную последовательность единиц в блоке (Test for the Longest Run of Ones in a Block)

В тесте определяется наиболее длинный ряд из единиц внутри блока длиной m. Целью является определить, действительно ли длина такого ряда соответствует ожиданиям длины самого протяжённого ряда единиц, в случае абсолютно случайной последовательности. Следует заметить, что из предположения о примерно одинаковой частоте появления единиц и нулей (тест № 1) следует, что точно такие же результаты данного теста будут получены при рассмотрении самого длинного ряда нулей. Поэтому измерения можно проводить только с единицами.

Тест рангов бинарных матриц (Binary Matrix Rank Test)

В тесте вычисляются ранги непересекающихся подматриц, построенных из исходной двоичной последовательности. Целью теста является проверка на линейную зависимость подстрок фиксированной длины, составляющих первоначальную последовательность.

Спектральный тест (Discrete Fourier Transform (Spectral) Test)

Основная идея теста заключается в оценке высоты пиков дискретного преобразования Фурье исходной последовательности. Цель — определить периодические свойства входной последовательности, например, близко расположенные друг к другу повторяющиеся участки. Тем самым это явно демонстрирует отклонения от случайного характера исследуемой последовательности. Идея состоит в том, чтобы число пиков, превышающих пороговое значение в 95 % по амплитуде, было значительно больше 5 %.

Тест на совпадение неперекрывающихся шаблонов (Non-overlapping Template Matching Test)

В тесте просиходит подсчёт количества заранее определенных шаблонов, найденных в исходной последовательности. Цель — выявление генераторов случайных или псевдослучайных чисел, формирующих слишком

часто заданные непериодические шаблоны. Как и в следующем тесте на совпадение перекрывающихся шаблонов, для поиска конкретных шаблонов длиной m бит используется окно также длиной m бит. Если шаблон не обнаружен, окно смещается на один бит. Если же шаблон найден, окно перемещается на бит, следующий за найденным шаблоном, и поиск продолжается дальше.

Тест на совпадение перекрывающихся шаблонов (Overlapping Template Matching Test)

Идея теста в подсчете количества заранее определенных шаблонов, найденных в исходной последовательности. Как и в предыдущем тесте, для поиска конкретных шаблонов длиной m бит используется окно также длиной m бит. Сам поиск производится аналогичным образом. Если шаблон не обнаружен, окно смещается на один бит. Разница между этим тестом и предыдущим заключается лишь в том, что если шаблон найден, окно перемещается только на бит вперед, после чего поиск продолжается дальше.

Универсальный статистический тест Маурера (Maurer's «Universal Statistical» Test)[80].

Происходит подсчет числа бит между одинаковыми шаблонами исходной последовательности. Цель теста — выяснить может ли данная последовательность быть значительно сжата без потерь информации. В случае, если это возможно сделать, то она не является истинно случайной. В ходе теста вычисляется значение вероятности p.

Тест на линейную сложность (Linear Complexity Test)

В тесте используется принцип работы линейного регистра сдвига с обратной связью (англ. Linear Feedback Shift Register, LFSR). Цель — выяснить, является ли входная последовательность достаточно сложной для того, чтобы считаться абсолютно случайной. Абсолютно случайные последовательности характеризуются длинными линейными регистрами сдвига с обратной связью. Если же такой регистр слишком короткий, то

предполагается, что последовательность не является в полной мере случайной. В ходе теста вычисляется значение вероятности p.

Тест на периодичность (Serial Test)

Тест сводится к подсчёту частоты всех возможных перекрываний шаблонов длины m бит на протяжении всей исходной последовательности. Целью является определение, действительно ли количество появлений 2m перекрывающихся шаблонов длиной m бит, приблизительно такое же, как в случае абсолютно случайной входной последовательности бит. Последняя, как известно, обладает однообразностью, то есть каждый шаблон длиной m бит появляется в последовательности с одинаковой вероятностью Стоит отметить, что при m=1 тест на периодичность переходит в частотный побитовый тест (№ 1).

Тест приблизительной энтропии (Approximate Entropy Test)

Как и в предыдущем тесте, в данном тесте акцент делается на подсчёте частоты всех возможных перекрываний шаблонов длины m бит на протяжении исходной последовательности битов. Цель теста — сравнить частоты перекрывания двух последовательных блоков исходной последовательности с длинами m и m+1 с частотами перекрывания аналогичных блоков в абсолютно случайной последовательности. *Тест кумулятивных сумм (Cumulative Sums (Cusum) Test)*

Суть теста в максимальном отклонении (от нуля) при произвольном обходе, определяемым кумулятивной суммой заданных (-1, +1) цифр в последовательности. Цель теста — определение того, является ли кумулятивная сумма частичных последовательностей, возникающих во входной последовательности, слишком большой или слишком маленькой по сравнению с ожидаемым поведением такой суммы для абсолютно случайной входной последовательности. Таким образом, кумулятивная сумма может рассматриваться как произвольный обход. Для случайной последовательности отклонения от произвольного обхода должны быть вблизи нуля. Для некоторых типов последовательностей, не являющихся в

полной мере случайными подобные отклонения от нуля при произвольном обходе будут достаточно существенными.

Тест на произвольные отклонения (Random Excursions Test)

В тесте просиходит подсчет числа циклов, имеющих k посещений при обходе кумулятивной суммы. Произвольный обход такой суммы начинается с сумм после последовательности (0,1),переведенной в последовательность (-1,+1) как в тесте №1. Цель данного теста — определить, отличается ли число посещений определенного состояния внутри цикла от аналогичного числа в случае абсолютно случайной входной последовательности. Фактически, данный тест есть набор, состоящий из восьми тестов, проводимых для каждого из восьми состояний цикла: −4, −3, −2, −1 и +1, +2, +3, +4. В каждом таком тесте принимается решение о степени случайности исходной последовательности в соответствии со следующим правилом: если вычисленное в ходе теста значение вероятности $p < 0,01$, то входная двоичная последовательность не является абсолютно случайной. В противном случае она носит случайный характер.

Другой тест на произвольные отклонения (Random Excursions Variant Test)

В этом тесте подсчитывается общее число посещений определенного состояния при произвольном обходе кумулятивной суммы. Целью является определение отклонений от ожидаемого числа посещений различных состояний при произвольном обходе. В действительности этот тест состоит из 18 тестов, проводимых для каждого состояния: −9, −8, …, −1 и +1, +2, …, +9. На каждом этапе делается вывод о случайности входной последовательности.

Результаты тестирования

По результатам 15 тестов каждый из представленных алгоритмов показал отличные друг от друга зависимости количества успешных тестирований от числа раундов. Каждый тест для обеспечения объективности эксперимента выполнялся на 10 различных ключах шифрования.

Для поиска минимального числа раундов, на которых обеспечиваются удовлетворительные статистические свойства шифртекста необходимо вычислить границу прохождения общего теста NIST.

В соответствии с критерием Пирсона вычислим значение хи-квадрат по формуле:

$$x^2 = \sum_{i=1}^{r} \frac{(v_i - np_i)^2}{np_i}, \quad (4)$$

где v – успешно пройденные тестирования, а np = 150 (15 тестов проводятся на 10 разных ключах шифрования). Экспериментально найдем, что при vi = 126 значение x2 = 3.841, что соответствует уровню значимости a = 0.05. Из этого следует, что при 126 пройденных тестах вероятность корректного определения случайности выходной последовательности шифра равна 95% [81].

Для иллюстрации результатов исследования дополнительно разработанный скрипт создаёт графики зависимости проведённых тестов от количества раундов, на которых тесты были пройдены. С полным кодом скрипта, выполняющего формирование графиков, можно ознакомиться в Приложении Ж. По результатам сформированных отчётов скриптом собирается информация о том, пройден ли тест. Графическое представление результатов тестирование хорошо иллюстрирует зависимости статистических свойств шифров от числа раундов.

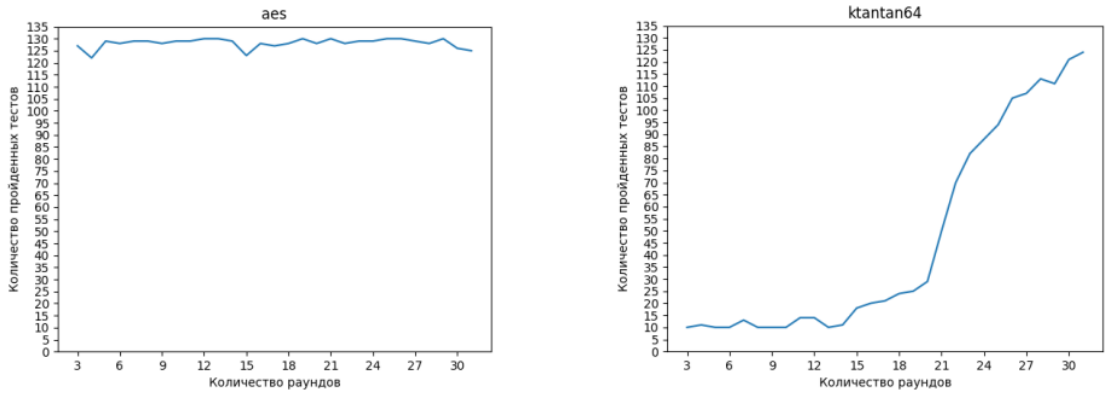


Рис. 21. Графики зависимостей количества пройденных тестов от числа раундов для алгоритмов AES и ktantan64.

Таблица 10 – Результаты тестирований NIST

Алгоритм	Full rounds	R_{min} NIST	$\%Rmin_R$
AES	10-14	3	21-30
CLEFIA	18-26	6	23-33
DESXL	8	6	75
HIGHT	32	10	31
IDEA	8.5	3	35
KATAN64	254	32	12
KLEIN	12-20	3	15-25
KTANTAN64	254	32	12
LED	32-48	4	8-12
Mcrypton	12	6	50
MIBS	32	3	9
NOEKEON	16	3	19
Piccolo	25-31	6	19-24
PRESENT	31	10	32
SEA	51	10	19
Simon	32-72	14	22-50
Speck	22-34	8	26-41
Twine	36	9	25
XTEA	32	3	9
Cast256	12	3	25
Twofish	16	3	19

Таблица 11 - Результаты тестирований NIST

Шифр	Rounds	R_{min}	%R
Anubis	12	3	25
Aria	12	3	25
Кузнечик	10	5	50
Mars	32	4	12
SM4	32	9	28
Threefish	72	3	4

Результаты проведённых тестов показывают, что параметр R_{min} может быть найден для каждого из представленных в настоящей работе итеративных блочных шифров. Доля минимального числа раундов каждого конкретного алгоритма варьируется от 4 до 75 процентов, однако с уверенностью можно заявить, что существует такая усеченная характеристика работы шифра, которая позволяет добиться удовлетворительных статистических свойств быстрее.

В таблице 12 приведены сводные результаты тестирований по рассмотренным в диссертационной работе методам.

Таблица 12 – результаты применения решающих правил, построенных согласно предлагаемому методу, к анализу генераторов псевдослучайных чисел на основе итеративных блочных шифров (жирным выделены шифры, для которых новые решающие правила позволяют достичь лучших результатов по сравнению с ранее опубликованными работами: либо увеличить R_{min}, либо уменьшить размер выборки)
Шифр – название шифра, на основе которого создан генератор. *R* – полное число раундов шифра. R_{min} *(NIST)* – минимальное число раундов, определенное тестами NIST (лучший из всех тестов по результатам экспериментов). R_{min} *BS* – минимальное число раундов определенное тестом «стопка книг». R_{min} $\chi 2$– минимальное число раундов определенное тестом «адаптивный критерий $\chi 2$», $R_{min(other)}$ – минимальное значение опубликованное другими авторами, R_{min} *MLSA* – минимальное число раундов определенное методом MLSA.

Шифр	***R***	***R_{min} (NIST)2^{27}***	***R_{min} BS (2^{24})***	***R_{min} $\chi 2$ (2^{24})***	***R_{min}(other)***	***R_{min} MLSA ($2^{21.9}$)***
CAST-128	***16***	***3***	***3***	***3***	***3(2^{31})***	***5***
IDEA	***8.5***	***3***	***2***	***2***	***2(2^{108})***	***2***
LBlock	***32***	***8***	***8***	***7***	***8(2^{26})***	***9***
Simon	***32-72***	***14***	***12***	***12***	***12(2^{36})***	***15***
Speck	***22-34***	***8***	***6***	***6***	***6(2^{31})***	***9***
RC5	***12***	***5***	***5***	***5***	***5(2^{25})***	***5***
RC6	***20***	***5***	***5***	***5***	***5(2^{29})***	***5***
AES	10-14	3	4	4	-	3
CLEFIA	18-26	6	6	6	-	5
DESXL	8	6	2	4	-	3
HIGHT	32	10	10	10	-	10
KATAN64	254	32	30	31	-	31
KLEIN	12-20	3	4	4	-	3

Шифр	***R***	R_{min} ***(NIST)*** 2^{27}	R_{min} ***BS*** (2^{24})	R_{min} $\chi 2$ (2^{24})	R_{min}***(other)***	R_{min} ***MLSA*** $(2^{21.9})$
KTANTAN64	254	32	30	31	-	31
LED	32-48	4	4	4	-	4
Mcrypton	12	6	6	3	-	5
MIBS	32	3	2	1	-	2
NOEKEON	16	3	2	2	-	2
Piccolo	25-31	6	6	5	-	5
PRESENT	31	10	9	8	-	8
SEA	51	10	10	10	-	10
Twine	36	9	9	9	-	9
XTEA	32	3	3	3	-	3

Таким образом, предлагаемый метод построения решающих правил показывает результаты, сопоставимые с известными статистическими тестами, используя при этом меньшую длину выборки (во всех экспериментах методом MLSA использована выборка $2^{21.9}$ бит), а для ряда шифров получены результаты, превосходящие ранее полученные (в таблице выделены жирным и серым цветом).

ВЫВОДЫ

Глава 3 диссертационной работы посвящена техническим аспектам разработки информационно-аналитической системы «УНИБЛОКС-2015». Проведен комплекс мер по подготовке программных реализаций итеративных блочных алгоритмов, а также предложено программное решение, позволяющее осуществлять корректное взаимодействие итеративных блочных шифров системы и статистических тестов.

Описана основная проблема, возникающая при проведении статистического анализа итеративных блочных шифров с помощью классических тестов, использующих методы математической статистики. Приведена общая информация о статистическом анализе. Для системы «УНИБЛОКС-2015» разработан перечень требований, предъявляемых к системе. Приводится описание используемых технологий программирования.

Приводится описание статистических тестов «стопка книг», «адаптивный криетрий хи-квадрат», тестов NIST.

Найдены параметры минимального числа раундов для ряда итеративных генераторов, основанных на блочных шифрах, при которых обеспечиваются удовлетворительные статистические свойства (R_{min}). Установлено, что для всех алгоритмов существует параметр R_{min}. Построены зависимости пройденных тестов NIST от числа раундов.

ЗАКЛЮЧЕНИЕ

В соответствии с поставленной целью и задачами исследования были получены следующие результаты.

1. Разработана информационно-аналитическая система для статистического анализа «НЕЙРОБЛОКС-2020», включающая более 25 алгоритмов блочного шифрования из существующих библиотек BLOC и CPPCRYPTO. В систему интегрировано 17 статистических тестов, разработаны инструменты, обеспечивающие взаимодействие тестов и шифров, а также инструментарий для проведения анализа и формирования отчётов (в том числе в графическом виде). Для всех предложенных в работе итеративных блочных шифров найден параметр R_{min} – минимальное число раундов шифрования, при котором обеспечиваются удовлетворительные статистические свойства полученной выходной последовательности итеративного генератора.

2. Предложен универсальный метод построения решающих правил на основе свёрточных нейронных сетей для анализа итеративных генераторов псевдослучайных чисел, в основе которого лежит использование технологий машинного обучения за счёт различения графических эквивалентов выходных последовательностей этих генераторов. На примере итеративных блочных шифров проведена апробация предлагаемого метода. Разработаны сопутствующие программные утилиты для работы предлагаемого алгоритма. Проанализированы и выбраны параметры работы модели нейронной сети Inception ResNet-v2, обеспечивающие оптимальный баланс точности анализа изображений и производительности для предлагаемого метода.

3. Предложено теоретическое обоснование эффективности предлагаемого метода. Экспериментально показано, что при попадании доли ошибок модели нейронной сети в рассчитанный доверительный интервал можно сделать вывод о статистических свойствах выходной последовательности как неотличимой от случайной. Эффективность

предлагаемого метода заключается в возможности использования существенно уменьшенной выборки по сравнению с использованием классических статистических тестов.

4. Сравнение полученных на примере блочных шифров результатов продемонстрировало, что новый метод позволил выявить отклонения от случайности на большем числе раундов у нескольких алгоритмов. Для большинства рассмотренных в настоящей работе алгоритмов метод снизил длину выборки для проведения атаки-различителя за счёт анализа всей выборки в целом, а не каждого конкретного значения последовательности. Полученные результаты минимального числа раундов шифрования, при котором обеспечиваются удовлетворительные статистические свойства, согласовываются с полученными ранее результатами посредством других методов статистического анализа.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1 Рябко Б.Я., Фионов А.Н. Основы современной криптографии и стеганографии // М.: Горячая линия-Телеком, 2010. 232 с.

2 Кяжин С.Н., Моисеев А.В. Криптография в облачных вычислениях: современное состояние и актуальные задачи // Безопасность информационных технологий. 2013. №3. С.83-86.

3 Пестунов А.И., Перов А.А., Пестунова Т.М. О некоторых направлениях научных исследований в области криптоанализа симметричных алгоритмов // Вестник НГУЭУ, 2015. с.280-298.

4 Словарь криптографических терминов / Под ред. Б.А. Погорелова и В.Н. Сачкова. - М.МЦНМО, 2006 - 94 с.

5 Гатченко Н.А., Исаев А.С., Яковлев А.Д. «Криптографическая защита информации» – СПб: НИУИТМО, 2012. – 142с

6 Шнайер Б. Прикладная криптография: Пер. с англ. - М: Триумф, 2002. - 610 с.

7 Rukhin A., Soto J., Nechvatal J., Smid M., Barker E., Leigh S., Levenson M., Vangel M., Banks D., Heckert A., Dray J., Vo S. / A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, Special Publication 800-22 Revision 1a, 2010, P.131.

8 Перов А.А., Пестунов А.И. Статистическое тестирование современных итеративных блочных шифров с помощью программной библиотеки "УНИБЛОКС-2015" // Инновации в жизнь. №2. 2016. с.89-97.

9 Knudsen L., Meier W. Correlations in RC6 // Proc. Fast Software Encryption-2001.

10 Жуков А.Е. Легковесная криптография. Часть 1. // Вопросы кибербезопасности №1. 2015. с.26-43.

11 Mathieu David "Lightweight Cryptography for Passive RFID Tags. 2011

12 Жуков А.Е. Легковесная криптография. Часть 2. // Вопросы кибербезопасности №2. 2015. с.2-10.

13 Шолле Франсуа. Глубокое обучение на Python. – СПб.: Питер, 2018. – 400 с.

14 Галимов Р.Г. Основы алгоритмов машинного обучения - обучение без учителя / Научно-практический электронный журнал "Аллея Науки" №14, 2017. с.807-809

15 Галимов Р.Г. Основы алгоритмов машинного обучения - обучение без учителя / Научно-практический электронный журнал "Аллея Науки" №14, 2017. с.810-817

16 He, K., Zhang, X., Ren, S., and Sun, J. Deep residuallearning for image recognition.CVPR, pp. 770–778,2016.

17 Терещенко С.Н. Проблема трудоустройства в эпоху искусственного интеллекта / International Journal of Advanced Studies, Vol.7, No 4-3, 2017. p.132-135.

18 Silver D., Huang A., Maddison C., Guez A., Sifre A., Driessche G., Schrittwieser J., Antonoglou I., Panneershelvam V., Lanctot M., Dieleman S., Grewe D., Nham J., Kalchbrenner N., Sutskever N., Lillicrap T., Leach M., Kavukcuoglu K., Graepel1 T., Hassabis D. Mastering the game of Go with deep neural networks and tree search / Nature, Vol 529, 2016. p.484-503.

19 Крутов Б.С, Августинович С.А., Цыганова А.А., Швалев А.Е., Сагалакова О.В., Кернякевич П. С. Машинное обучение в банковской сфере / Вестник современных исследований, 2018. с.490-492

20 He, K., Zhang, X., Ren, S., and Sun, J. Deep residuallearning for image recognition.CVPR, pp. 770–778,2016.

21 Попова Н.А., Назаров М.А., Власов М.В. Решение задачи распознавания лиц с использованием алгоритмов машинного обучения / Моделирование, оптимизация и информационные технологии, Научный журнал. Том 6, №1. с.408-415

22 Турыгина В.Ф., Форд В.Ф., Матвевнина А.И. Применение машинного обучения в кибер-безопасности / Научный альманах, №6-1(32), 2017. с.405-408

23 Abu-Nimeh S. , Nappa D., Wang X., Nair S., ― A comparison of machine learning techniques for phishing detection,‖ in Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit, ser. eCrime ’07. New York, NY, USA: ACM, 2007, pp. 60–69.

24 Anti-Phishing Working Group, “Phishing and Fraud solutions”. [Online].

25 M. Wu, R. C. Miller, and S. L. Garnkel, “Do security toolbars actually prevent phishing attacks?” in Proceedings of the SIGCHI conference on Human Factors in computing systems, 2006.

26 Cranor L. F., Egelman S., Hong J., Zhang Y., “Phinding phish: An evaluation of anti-phishing toolbars”, Technical Report CMU-CyLab-06-018, CMU, November 2006.

27 Zhuang W., Ye Y., Chen Y., Li T., “Ensemble Clustering for Internet Security Applications”, in IEEE xplore, December 17, 2012.

28 Lu N., Mabu S., Wang T., Hirasawa K., “An Efficient Class Association Rule-Pruning Method for Unified Intrusion Detection System using Genetic Algorithm”, in IEEJ Transactions on Electrical and Electronic Engineering, Vol. 8, Issue 2, pp. 164 – 172, January 2, 2013.

29 Knowledge Discovery and Data Mining group, “KDD cup 1999”. [Online]. Available: http://www.kdd.org/kddcup/index.php. [Accessed: March 3, 2013].

30 Subbulakshmi T., Shalinie S. M., Ramamoorthi A., “Detection and Classification of DDoS Attacks using Machine Learning Algorithms”, European Journal of Scientific Research, ISSN 1450-216X, Volume 47, No. 3, pp. 334 – 346, 2010.

31 Sedjelmaci H., Feham M., “Novel Hybrid Intrusion Detection System for Clustered Wireless Sensor Network”, International Journal of Network Security & Its Applications (IJNSA), Vol.3, No.4, July 2011.

32 Revett K. et al., “A machine learning approach to keystroke dynamics based user authentication”, International Journal of Electronic Security and Digital Forensics, Vol. 1, No. 1, 2007.

33 Chellapilla K. and Simard P. Y., “Using Machine Learning to Break Visual Human Interaction Proofs (HIPs)”, in Advances in Neural Information Processing Systems 17, pp. 265-272, 2005

34 Simard P.Y., Steinkraus D, Platt J, (2003) “Best Practice for Convolutional Neural Networks Applied to Visual Document Analysis,” in International Conference on Document Analysis and Recognition(ICDAR), pp. 958-962, IEEE Computer Society, Los Alamitos.

35 Rivest R.L., “Cryptography and machine learning,” inAdvances in Cryptology — ASIACRYPT ’91(H. Imai, R. L. Rivest, and T. Matsumoto, eds.),(Berlin, Heidelberg), pp. 427–439, Springer Berlin Heidelberg, 1993

36 Yu W. and Cao J., “Cryptography based on delayed chaotic neural networks”, Physics Letters A, Vol. 356, Issues 4–5, pp. 333-338, ISSN 0375-9601, August 14, 2006.

37 Kinzel W. and Kanter I., “Neural Cryptography”, in Proceedings of the 9th International Conference on Neural Information Processing, Vol. 3, pp. 1351-1354, November 18-22, 2002.

38 Монарёв В.А., Пестунов А.И. Эффективное обнаружение стеганографически скрытой информации посредством интегрального классификатора на основе сжатия данных / Прикладная дискретная математика, №40, 2018. с.59-71

39 Mohammed M. Alani. Applications of Machine Learning in Cryptography: A Survey. 1, 1, 2019, 8 p.

40 Weissbart L., Picek S., Batina L. One trace is all it takes: Machine Learning-based Side-channel Attack on EdDSA / Cryptology ePrint Archive: Report 2019/358, 2019. 18 p.

41 Gohr A. Improving Attacks on Round-Reduced Speck32/64 Using Deep Learning. In: Boldyreva A., Micciancio D. (eds) Advances in Cryptology – CRYPTO 2019. CRYPTO 2019. Lecture Notes in Computer Science, vol 11693. Springer, Cham

42 Lerman L., Bontempi G., Markowitch O. Power analysis attack: an approach based on machine learning / International Journal of Applied Cryptography, Volume 3 Issue 2, June 2014, p.97-115

43 Пестунов А.И. Предварительная оценка минимального числа раундов легковесных шифров для обеспечения их удовлетворительных статистических свойств // Прикладная дискретная математика.Приложение. 2015. С.66-69.

44 Рябко Б.Я., Фионов А.Н. Основы современной криптографии и стеганографии М.: Горячая линия-Телеком, 2010. — 232 с.

45 Слеповичев И.И. Генераторы псевдослучайных чисел / Учебное пособие – 118 с.

46 Пестунов А.И., Перов А.А. Программная библиотека для статистического анализа итеративных блочных шифров // Информационное противодействие угрозам терроризма. 2015. №24. С.197-202.

47 Wu W., Zhang L. LBlock: A Lightweight Block Cipher //ACNS'11 Proceedings of the 9th international conference on applied cryptography and network security. 2011. Pp. 327-344.

48 Bogdanov A., Knudsen L.R., Leander G., Paar C., Poschmann A., Robshaw M., Seurin Y., Vikkelsoe C. Present: An ultra-lightweight block cipher //CHES '07 Proceedings of the 9th international workshop on Cryptographic Hardware and Embedded Systems. 2007. Pp.450-466.

49 Sekar G., Mouha N., Velichkov V., Preneel B. Meet-in-the-middle attacks on reduced-round XTEA. CT-RSA'11 Proceedings of the 11th international conference on Topics in cryptology: CT-RSA 2011. 2011. Pp.250-267.

50 Suzaki T., Minematsu K., Morioka S., Kobayashi E. TWINE: A lightweight, versatile block cipher. 2011. [Электронный документ] URL: http://www.nec.co.jp/rd/media/code/research/images/twine_LC11.pdf

50 Beaulieu R., Shors D., Smith J., Treatman-Clark S., Weeks B., Wingers L. The SIMON and SPECK lightweight block ciphers // DAC '13 Proceedings of the 52nd Annual Design Automation Conference. 2013. Pp.175 - 220.

52 Shirai T., Shibutani K.,Akishita T., Moriai S., Iwata T. The 128-bit blockcipher CLEFIA // FSE'07 Proceedings of the 14th international conference on fast software encryption. 2007. Pp. 181-195.

53 Sasarak C., Sridharan S. HIGHT: a new block cipher suitable for low-resource device // CHES'06 Proceedings of the 8th international conference on Cryptographic Hardware and Embedded Systems.2006. Pp.46-59.

54 Shibutani K.,Isobe T., Hiwatari H., Mitsuda A., Akishita T., Shirai T. Piccolo: an ultra-lightweight blockcipher // CHES'11 Proceedings of the 13th international conference on cryptographic hardware and embedded systems. 2011. Pp.342-357.

55 Gong Z., Nikova S., Wei Law Y. KLEIN: A new family of lightweight block ciphers // RFIDSec'11 Proceedings of the 7th international conference on RFID Security and Privacy. 2011. Pp. 1-18.

56 Sanku V. Implementation and performance analysis of Skipjack & Rijndael Algorithms. 2009. [Электронный документ] URL: http://teal.gmu.edu/courses/ECE646/project/slides_2001/sanku.pdf

57 Hoon Lim C., Korkishko T. mCrypton – a lightweight block cipher for security of low-cost RFID tags and sensors //WISA'05 Proceedings of the 6th international conference on Information Security Applications. 2005. Pp. 243-258.

58 Guo J., Peyrin T., Poschmann A., Robshaw M. The LED Block Cipher //CHES'11 Proceedings of the 13th international conference on Cryptographic hardware and embedded systems. 2011. Pp. 326-341.

59 Daemen J., Peeters M., Van Assche G., Rijmen V. Nessie Proposal: NOEKEON. 2000. [Электронный документ] URL: http://fallbackstatus.com/2b6b453a58c6646b.html

60 Mace F., Standaert F.C., Quisquarter J.J.ASIC Implementations of the block cipher SEA for constrained applications // In RFID Security — RFIDsec 2007. Pp.103-114.

61 Xiaoshuang Ma, Lei Hu, Siwei Sun, Kexin Qiao, Jinyong S. Tighter Security Bound of MIBS Block Cipher against Differential Attack / Network and System Security: 8th International Conference, NSS 2014, Xi'an, China, October 15-17, 2014, Proceedings. pp.518-525.

62 Junod P. IDEA new attacks againgst reduced-round versions //Proceedings of Fast Software Encryption 2005, Lecture Notes in Computer Science 3557. 2005. Pp. 384–397.

63 Advanced Encryption Standart (AES) Federal Information Processing Standards publication 197. 2001. 51p.

64 Data Encryption Standart U.S. Department of commerce / National Institute of Standarts and Technology, 1999, 27 p.

65 CanniereC., Dunkelman O.,Knezevi M. KATAN & KTANTAN — A family of small and efficient hardware-oriented block ciphers// CHES '09 Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems. 2009. Pp. 272-288.

66 Barreto P., Rijmen V. The Anubis Block Cipher, Nessie Proposal. 26p.

67 Kwon D., Kim J., Sung S.H. New block cipher: ARIA, Lecture Notes in Computer Science, 2003, p.432-435

68 ГОСТ 34.12-2015 Информационная технология Криптографическая защита информации, Блочные шифры, 2015, 15 с.

69 Burwick C., Coppersmith D., D'Avignon E., Gennaro R., Halevi S., Jutla C., Matyas S., O'Connor L., Peyravian M., Safford D., Zunic N. The MARS Encryption Algorithm, 1999, 12 p.

70 Diffie W., Ledin G. SMS4 Encryption Algorithm for Wireless Networks, 2008, 6 p.

71 At N., Beuchat J.-L. Compact Implementation of Threefish and Skein on FPGA / New Technologies, Mobility and Security, 2012 5th International Conference

72 Хэлм Р., Гамма Э., Джонсон Р., Влисдес Дж. Приёмы объектно-ориентированного проектирования. Паттерны Проектирования. Питер, 2001. - 368 с.

73 Паттерн Стратегия - «Strategy» :: ХабраХабр [Электронный ресурс] URL:http://habrahabr.ru/post/84643/

74 Рябко Б.Я., Пестунов А.И. «Стопка книг» как новый статистический тест для случайных чисел // Проблемы передачи информации. 2004. Т. 40,№ 1. С.73-78.

75 Рябко Б.Я., Стогниенко В.С., Шокин Ю.И. Адаптивный критерий $\chi 2$ для различения близких гипотез при большом числе классов и его применение к некоторым задачам криптографии // Проблемы передачи информации. 2003. Т. 39,№ 2. С.53-62.

76 Перов А.А. Применение статистических тестов NIST для анализа выходных последовательностей блочных шифров // Научный вестник НГТУ. – 2019. – № 3 (76). – С. 87–96.

77 FIPS 140-1, Security Requirements for Cryptographic Modules, Federal Information Processing Standards Publication 140-1. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, 1994.

78 FIPS 180-1, Secure Hash Standard, Federal Information Processing Standards Publication 180-1. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, April 17, 1995.

79 FIPS 186, Digital Signature Standard (DSS), Federal Information Processing Standards Publication 186. U.S. Department of Commerce/NIST, National Technical Information Service, Springfield, VA, May 19, 1994.

80 U. Maurer, “A Universal Statistical Test for Random Bit Generators,” Journal of Cryptology. Vol. 5, No. 2, 1992, pp. 89-105

81 Ивченко Г.И., Медведев Ю.И. Введение в математическую статистику: Учебник. М: Издатеьство ЛКИ, 2010. - 600 с.

82 Krizhevsky A., Sutskever I., Hinton G.} // ImageNet Classification with Deep Convolutional Neural Networks, NIPS 2012.

83 J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. InCVPR,2009.

84 M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Is object localizationfor free? – weakly-supervised learning with convolutional neuralnetworks. InCVPR, 2015

85 M. Ren, R. Kiros, and R. Zemel. Exploring models and data forimage question answering. InNIPS, 2015

86 R. R. Selvaraju, S. Lee, Y. Shen, H. Jin, S. Ghosh, L. Heck, D. Batra,and D. Parikh. Taking a hint: Leveraging explanations to makevision and language models more grounded. InProceedings of theInternational Conference on Computer Vision (ICCV), 2019

87 Abramowitz M. and Stegun I., Handbook of Mathematical Functions, Applied Mathematics Series. Vol. 55, Washington: National Bureau of Standards, 1964; reprinted 1968 by Dover Publications, New York

88 George, D., Shen, H., Huerta, E.: Deep transfer learning: a new deep learning glitch classification method for advanced LIGO. arXiv preprint arXiv:1706.07446 (2017)

89 Dai, W., Yang, Q., Xue, G.R., Yu, Y.: Boosting for transfer learning. In: Proceedings of the 24th International Conference on Machine Learning, pp. 193–200. ACM (2007)

90 Li, N., Hao, H., Gu, Q., Wang, D., Hu, X.: A transfer learning method for automatic identification of sandstone microscopic images. Comput. Geosci. 103, 111–121 (2017)

91 Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., Wojna, Z.: Rethinking the inception architecture for computer vision. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2818–2826 (2016)

92 Endert, A. & Ribarsky, W. & Turkay, Cagatay & Wong, B.L. & Nabney, Ian & Díaz Blanco, Ignacio & Rossi, Fabrice. (2017). The State of the Art in Integrating Machine Learning into Visual Analytics: Integrating Machine Learning into Visual Analytics. Computer Graphics Forum.

93 Huang, G., Liu, Z., Weinberger, K.Q., van der Maaten, L.: Densely connected convolutional networks. In: CVPR, pp. 4700–4708. IEEE, 2017.

94 Chollet F.: Xception: Deep learning with depthwise separable convolutions. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1251–1258, 2017.

95 EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Mingxing Tan. , Quoc V. Le. Proceedings of the 36th International Conference on Machine Learning, 2019.

96 Szegedy C, Ioffe S, Vanhoucke V, Alemi AA (2017) Inception-v4, inception-resnet and the impact of residual connections on learning. In: Thirty-first AAAI conference on artificial intelligence, 2017.

97 Perov A. Using Machine Learning Technologies for Carrying out Statistical Analysis of Block Ciphers // 2019 International Multi-Conference On Engineering Computer And Information Sciences (SIBIRCON), p.851-854, 2019.

98 Перов А.А. Об использовании технологий машинного обучения для проверки статистических свойств симметричных криптографических алгоритмов //Прикладная дискретная математика. Приложение. - 2019 - №12. - С. 232-235.

99 Перов А.А. Анализ возможностей применения технологий искусственного интеллекта к задачам статистического анализа блочных шифров // Тезисы докладов XX Всероссийской конференции молодых учёных по математическому моделированию и информационным технологиям YM-2019. - 2019. - с.72-73.

100 Перов А.А. О возможности применения свёрточных нейронных сетей к построению универсальных атак на итеративные блочные шифры / А.А. Перов, А.И. Пестунов // Прикладная дискретная математика – 2020. – №3 (49) – С. 46-57. (индексируется в Scopus и Web of Science)

101 Перов А.А. Построение различителей для итеративных блочных шифров на основе нейронных сетей/ А.А. Перов, А.И. Пестунов // Прикладная дискретная математика. Приложение – 2020 – №13 – С. 54-56.

102 ГОСТ Р 50.1.028-2001 "Информационные технологии поддержки жизненного цикла продукции. Методология функционального моделирования"

103 Севастьянов Б.А. Курс теории вероятностей и математической статистики М.: Наука. Гл. ред. физ.-мат. лит., 1982.— 256 с.

104 Bernardi, Marcello and Khouzani, M. and Malacaria, Pasquale. Pseudo-Random Number Generation Using Generative Adversarial Networks, Advances in Mathematics and Bio-mathematics, pp.191 - 200, 2019.

105 Сикорский О.С. Обзор свёрточных нейронных сетей для задачи классификации изображений / Новые информационные технологии в автоматизированных системах. - 2017, С.37-42

106 Пестунов А.И. Унифицированная программная библиотека для статистического анализа итеративных блочных шифров «Униблокс-2015» / А.И. Пестунов, А.А. Перов, Т.М. Пестунова // М.: Роспатент. Свидетельство о государственной регистрации программы для ЭВМ №2015662092, от 17.11.2015.

107 Перов А.А. Программный комплекс для статистического анализа и оценки стойкости итеративных блочных шифров при помощи сверточных

нейронных сетей «Нейроблокс-2020» / Перов А.А., Пестунов А.И. // М.: Роспатент. Свидетельство о государственной регистрации программы для ЭВМ №2020618202, от 07.08.2020.

ПРИЛОЖЕНИЕ А

Программный код файла utils.h

```
#include <stdlib.h>

#ifndef BLOCK_CIPHERS_UTILS
#define BLOCK_CIPHERS_UTILS

#define ROTATE_LEFT_8(x,n) ( ((x) << (n)) | ((x) >> (8-(n))) )
#define ROTATE_LEFT_16(x,n) ( ((x) << (n)) | ((x) >> (16-(n))) )
#define ROTATE_LEFT_32(x,n) ( ((x) << (n)) | ((x) >> (32-(n))) )
#define ROTATE_LEFT_64(x,n) ( ((x) << (n)) | ((x) >> (64-(n))) )
#define ROTATE_RIGHT_8(x,n) ( ((x) >> (n)) | ((x) << (8-(n))) )
#define ROTATE_RIGHT_16(x,n) ( ((x) >> (n)) | ((x) << (16-(n))) )
#define ROTATE_RIGHT_32(x,n) ( ((x) >> (n)) | ((x) << (32-(n))) )
#define ROTATE_RIGHT_64(x,n) ( ((x) >> (n)) | ((x) << (64-(n))) )

typedef unsigned char  u8;
typedef unsigned short u16;
typedef unsigned int u32;
typedef unsigned long long u64;

typedef int s32;
typedef char s8;

//Join short words
void _square4to32(u8 square8[4][4], u32* text32) {
      text32[0] = square8[0][0] & 0x0F;
      text32[0] <<= 4;
      text32[0] += square8[0][1] & 0x0F;
      text32[0] <<= 4;
      text32[0] += square8[0][2] & 0x0F;
      text32[0] <<= 4;
      text32[0] += square8[0][3] & 0x0F;
      text32[0] <<= 4;
      text32[0] += square8[1][0] & 0x0F;
      text32[0] <<= 4;
      text32[0] += square8[1][1] & 0x0F;
      text32[0] <<= 4;
      text32[0] += square8[1][2] & 0x0F;
      text32[0] <<= 4;
      text32[0] += square8[1][3] & 0x0F;

      text32[1] = square8[2][0] & 0x0F;
      text32[1] <<= 4;
      text32[1] += square8[2][1] & 0x0F;
      text32[1] <<= 4;
      text32[1] += square8[2][2] & 0x0F;
      text32[1] <<= 4;
      text32[1] += square8[2][3] & 0x0F;
      text32[1] <<= 4;
      text32[1] += square8[3][0] & 0x0F;
      text32[1] <<= 4;
      text32[1] += square8[3][1] & 0x0F;
      text32[1] <<= 4;
      text32[1] += square8[3][2] & 0x0F;
      text32[1] <<= 4;
      text32[1] += square8[3][3] & 0x0F;
}

void _32to64(u64* text64, u32* text32, int n_words64) {
```

```
	for (int i = 0; i < n_words64; i++) {
		text64[i] = text32[2 * i];
		text64[i] <<= 32;
		text64[i] += text32[2 * i + 1];
	}
}

void _16to32(u16* text16, u32* text32, int n_words32) {
	for (int i = 0; i < n_words32; i++) {
		text32[i] = text16[2 * i];
		text32[i] <<= 16;
		text32[i] += text16[2 * i + 1];
	}
}

void _8to32(u8* text8, u32* text32, int n_words32) {
	for (int i = 0; i < n_words32; i++) {
		text32[i] = text8[4 * i];
		text32[i] <<= 8;
		text32[i] += text8[4 * i + 1];
		text32[i] <<= 8;
		text32[i] += text8[4 * i + 2];
		text32[i] <<= 8;
		text32[i] += text8[4 * i + 3];
	}
}

void _4to32(u8* text4, u32* text32, int n_words32) {
	for (int i = 0; i < n_words32; i++) {
		text32[i] = text4[8 * i] & 0x0F;
		text32[i] <<= 4;
		text32[i] += text4[8 * i + 1] & 0x0F;
		text32[i] <<= 4;
		text32[i] += text4[8 * i + 2] & 0x0F;
		text32[i] <<= 4;
		text32[i] += text4[8 * i + 3] & 0x0F;
		text32[i] <<= 4;
		text32[i] += text4[8 * i + 4] & 0x0F;
		text32[i] <<= 4;
		text32[i] += text4[8 * i + 5] & 0x0F;
		text32[i] <<= 4;
		text32[i] += text4[8 * i + 6] & 0x0F;
		text32[i] <<= 4;
		text32[i] += text4[8 * i + 7] & 0x0F;

	}
}

//Split long words
void _64to32(u64* text64, u32* text32, int n_words64) {
	for (int i = 0; i < n_words64; i++) {
		text32[2 * i] = (text64[i] >> 32) & 0xFFFFFFFF;
		text32[2 * i + 1] = text64[i] & 0xFFFFFFFF;
	}
}

void _32to16(u16* text16, u32* text32, int n_words32) {
	for (int i = 0; i < n_words32; i++) {
		text16[2 * i] = (text32[i] >> 16) & 0x0000FFFF;
		text16[2 * i + 1] = text32[i] & 0x0000FFFF;
	}
}
```

```
void _32to8(u8* text8, u32* text32, int n_words32) {
	for (int i = 0; i < n_words32; i++) {
		text8[4 * i] = (text32[i] >> 24) & 0x000000FF;
		text8[4 * i + 1] = (text32[i] >> 16) & 0x000000FF;
		text8[4 * i + 2] = (text32[i] >> 8) & 0x000000FF;
		text8[4 * i + 3] = text32[i] & 0x000000FF;
	}
}

void _32to4(u8* text4, u32* text32, int n_words32) {
	for (int i = 0; i < n_words32; i++) {
		text4[8 * i] = (text32[i] >> 28) & 0x0F;
		text4[8 * i + 1] = (text32[i] >> 24) & 0x0F;
		text4[8 * i + 2] = (text32[i] >> 20) & 0x0F;
		text4[8 * i + 3] = (text32[i] >> 16) & 0x0F;
		text4[8 * i + 4] = (text32[i] >> 12) & 0x0F;
		text4[8 * i + 5] = (text32[i] >> 8) & 0x0F;
		text4[8 * i + 6] = (text32[i] >> 4) & 0x0F;
		text4[8 * i + 7] = text32[i] & 0x0F;
	}

}

void _32to4square(u8 text4[4][4], u32* text32) {
	text4[0][0] = (text32[0] >> 28) & 0x0F;
	text4[0][1] = (text32[0] >> 24) & 0x0F;
	text4[0][2] = (text32[0] >> 20) & 0x0F;
	text4[0][3] = (text32[0] >> 16) & 0x0F;
	text4[1][0] = (text32[0] >> 12) & 0x0F;
	text4[1][1] = (text32[0] >> 8) & 0x0F;
	text4[1][2] = (text32[0] >> 4) & 0x0F;
	text4[1][3] = text32[0] & 0x0F;

	text4[2][0] = (text32[1] >> 28) & 0x0F;
	text4[2][1] = (text32[1] >> 24) & 0x0F;
	text4[2][2] = (text32[1] >> 20) & 0x0F;
	text4[2][3] = (text32[1] >> 16) & 0x0F;
	text4[3][0] = (text32[1] >> 12) & 0x0F;
	text4[3][1] = (text32[1] >> 8) & 0x0F;
	text4[3][2] = (text32[1] >> 4) & 0x0F;
	text4[3][3] = text32[1] & 0x0F;

}

#endif
```

ПРИЛОЖЕНИЕ Б

Программный код файла blockciphers.h

```
#include "lblock.h"
#include "present.h"
#include "xtea.h"
#include "twine.h"
#include "keeloq.h"
#include "speck.h"
#include "simon.h"
#include "clefia.h"
#include "hight.h"
#include "piccolo.h"
#include "klein.h"
#include "skipjack.h"
#include "rc5.h"
#include "mcrypton.h"
#include "led.h"
#include "noekeon.h"
#include "sea.h"
#include "mibs.h"
#include "idea.h"
#include "aes.h"
#include "desxl.h"
#include "katan32.h"
#include "katan64.h"
#include "ktantan32.h"
#include "ktantan64.h"

enum Ciphers {
	LBLOCK,
	PRESENT,
	XTEA,
	TWINE,
	//KEELOQ,
	SPECK,
	SIMON,
	CLEFIA,
	HIGHT,
	PICCOLO,
	KLEIN,
	SKIPJACK,
	//RC5,
	MCRYPTON,
	LED,
	NOEKEON,
	SEA,
	MIBS,
	IDEA,
	AES,
	DESXL,
	//KATAN32,
	KATAN64,
	//KTANTAN32,
	KTANTAN64
};

void(*encrypt)(u32* text, int rounds);
```

```
void(*resetKey)();
char*(*id)();
void(*runTests)();

void setCipher(Ciphers cipher) {
      if (cipher == LBLOCK) {
            encrypt = Lblock::encrypt;
            resetKey = Lblock::resetKey;
            id = Lblock::id;
            runTests = Lblock::runTests;
      }
      else if (cipher == PRESENT) {
            encrypt = Present::encrypt;
            resetKey = Present::resetKey;
            id = Present::id;
            runTests = Present::runTests;
      }
      else if (cipher == XTEA) {
            encrypt = Xtea::encrypt;
            resetKey = Xtea::resetKey;
            id = Xtea::id;
            runTests = Xtea::runTests;
      }
      else if (cipher == TWINE) {
            encrypt = Twine::encrypt;
            resetKey = Twine::resetKey;
            id = Twine::id;
            runTests = Twine::runTests;
      }

      else if (cipher == SPECK) {
            encrypt = Speck::encrypt;
            resetKey = Speck::resetKey;
            id = Speck::id;
            runTests = Speck::runTests;
      }
      else if (cipher == SIMON) {
            encrypt = Simon::encrypt;
            resetKey = Simon::resetKey;
            id = Simon::id;
            runTests = Simon::runTests;
      }
      else if (cipher == CLEFIA) {
            encrypt = Clefia::encrypt;
            resetKey = Clefia::resetKey;
            id = Clefia::id;
            runTests = Clefia::runTests;
      }
      else if (cipher == HIGHT) {
            encrypt = Hight::encrypt;
            resetKey = Hight::resetKey;
            id = Hight::id;
            runTests = Hight::runTests;
      }
      else if (cipher == PICCOLO) {
            encrypt = Piccolo::encrypt;
            resetKey = Piccolo::resetKey;
            id = Piccolo::id;
            runTests = Piccolo::runTests;
      }
      else if (cipher == KLEIN) {
            encrypt = Klein::encrypt;
            resetKey = Klein::resetKey;
```

```
      id = Klein::id;
      runTests = Klein::runTests;
}
else if (cipher == SKIPJACK) {
      encrypt = Skipjack::encrypt;
      resetKey = Skipjack::resetKey;
      id = Skipjack::id;
      runTests = Skipjack::runTests;
}
else if (cipher == RC5) {
encrypt = Rc5::encrypt;
resetKey = Rc5::resetKey;
id = Rc5::id;
runTests = Rc5::runTests;
}
else if (cipher == MCRYPTON) {
      encrypt = Mcrypton::encrypt;
      resetKey = Mcrypton::resetKey;
      id = Mcrypton::id;
      runTests = Mcrypton::runTests;
}
else if (cipher == LED) {
      encrypt = Led::encrypt;
      resetKey = Led::resetKey;
      id = Led::id;
      runTests = Led::runTests;
}
else if (cipher == NOEKEON) {
      encrypt = Noekeon::encrypt;
      resetKey = Noekeon::resetKey;
      id = Noekeon::id;
      runTests = Noekeon::runTests;
}
else if (cipher == SEA) {
      encrypt = Sea::encrypt;
      resetKey = Sea::resetKey;
      id = Sea::id;
      runTests = Sea::runTests;
}
else if (cipher == MIBS) {
      encrypt = Mibs::encrypt;
      resetKey = Mibs::resetKey;
      id = Mibs::id;
      runTests = Mibs::runTests;
}
else if (cipher == IDEA) {
      encrypt = Idea::encrypt;
      resetKey = Idea::resetKey;
      id = Idea::id;
      runTests = Idea::runTests;
}
else if (cipher == AES) {
      encrypt = Aes::encrypt;
      resetKey = Aes::resetKey;
      id = Aes::id;
      runTests = Aes::runTests;
}
else if (cipher == DESXL) {
      encrypt = Desxl::encrypt;
      resetKey = Desxl::resetKey;
      id = Desxl::id;
}
```

```
else if (cipher == KATAN64) {
      encrypt = Katan64::encrypt;
      resetKey = Katan64::resetKey;
      id = Katan64::id;
}

else if (cipher == KTANTAN64) {
      encrypt = Ktantan64::encrypt;
      resetKey = Ktantan64::resetKey;
      id = Ktantan64::id;
}
```

ПРИЛОЖЕНИЕ В

Программный код теста «стопка книг»

```
#include "stdafx.h"
#include "blockciphers.h"
#include "BookStack32.h"
#include <iostream>
#include <string>
#include <ctime>
using namespace std;
int main(int argc, char **argv)
{

	srand(time(0));

	u32 blk[4];

	//Test
	unsignedinti;
	keyquo = 10; //Количество случайных ключей, для которых осуществляется
проверка

	int kol = 0;

	FILE* rfile = fopen("resultlk24 (s=32, 30).txt", "w");
	FILE* rsfile = fopen("resultlk24 (s=32, 30)_sum.txt", "w");
	Init(18, 24, 32);
	fprintf(rfile, "k=%u n=%u s=%u keyquo=%d\n", k1, n1, s1, keyquo);
	printf("k=%u n=%u s=%u keyquo=%d\n", k1, n1, s1, keyquo);

	fprintf(rfile, "%10s %10s %10s %10s\n", "Cipher", "perc05", "perc01",
"avx2");
	printf("%10s %10s %10s %10s\n", "Cipher", "perc05", "perc01", "avx2");

	for (int cip = LBLOCK; cip <= KTANTAN64; cip++) {
		setCipher((Ciphers)cip);
		fprintf(rsfile, "%c%c%c ", id()[0], id()[1], id()[2]);
	}
	fprintf(rsfile, "\n");

	int rnds = 0;
	do {

		//Инициализация теста 2^{18} - размер верхней части стопки книг,
		//2^{18} - размер тестируемых выборок
		//2^{32} - размер алфавита, т.е. 32 - размер тестируемых слов
		kol = 0;
		rnds++;
		fprintf(rfile, "ROUNDS=%d\n", rnds);
		printf("ROUNDS=%d\n", rnds);

		for (int cip = LBLOCK; cip <= KTANTAN64; cip++) {

			setCipher((Ciphers)cip);
			fprintf(rfile, "%10s: ", id());
			printf("%10s: ", id());

			//Start
			for (keyn = 0; keyn < keyquo; keyn++) {
				resetKey();
				nu = 0;
```

```
                        for (i = 0; i < N1; i++){
                              blk[0] = 0;
                              blk[1] = i;
                              blk[2] = 0;
                              blk[3] = 0;
                              encrypt(blk, rnds); //шифрование блока
                              Search(blk[0] & 0x0000FFFF); nu = nu + is;
//проверка "попало ли слово в верхнюю часть стопки книг"
                              Search(blk[0] & 0x00FFFFFF); nu = nu + is;
//проверка "попало ли слово в верхнюю часть стопки книг"
                              Search(blk[0]); nu = nu + is; //проверка "попало
ли слово в верхнюю часть стопки книг"
                              printf("%08x\n", state);
                              Search(data[1]); nu = nu + is;
                              Search(data[2]); nu = nu + is;
                              Search(data[3]); nu = nu + is;
                              //блок шифра RC6 имеет длину 128 битов, поэтому
в нем 4 таких слова
                        }
                        CalcChi(1); //Вычисление величины хи-квадрат для
обработанной выборки
                  }
                  Stats st = getStats();
                  fprintf(rfile, "%10.0f %10.0f %10.2f", st.perc05, st.perc01,
st.avx2);
                  printf("%10.0f %10.0f %10.2f", st.perc05, st.perc01,
st.avx2);

                  if (st.perc01 <= 10 && st.perc05 <= 20) {
                        fprintf(rfile, " --- NO\n");
                        printf(" --- NO\n");
                        fprintf(rsfile, "%3s ", "NO");
                  }
                  else {
                        fprintf(rfile, " +++ YES\n");
                        printf(" +++ YES\n");
                        fprintf(rsfile, "%3s ", "YES");
                        kol++;
                  }

                  //вывод статистики по всем выборкам

            }
            fprintf(rsfile, "\n");
            fprintf(rfile, "KOL=%d\n", kol);
            printf("KOL=%d\n", kol);
            fprintf(rfile, "-----------------------------------------------\n",
kol);
            printf("-----------------------------------------------\n", kol);

      } while (rnds <= 30);

      fclose(rfile);
      fclose(rsfile);

      system("pause");
      return 0;
}
```

ПРИЛОЖЕНИЕ Г

Программный код файла Bookstack.h

```
#include <stdio.h>
#include <stdlib.h>

unsigned int nu;
int u01 = 0, u05 = 0, keyn, keyquo;
double avx2, sumx2 = 0, x2, perc05, perc01;

struct Element
{
	unsigned int value;
	Element * right;
	Element * prev;
	Element * next;
	Element(unsigned int x)
	{
		value = x; right = NULL; prev = NULL; next = NULL;
	}
};
typedef Element*LINK;
LINK first, last;
int is;
LINK* ht;
int k1, n1, s1;
unsigned int K1, N1;
//Power*********************************
unsigned int Power(unsigned int x, int y){
	unsigned int res = 1; int i;
	for (i = 1; i <= y; i++) res = x*res;
	return res;
}
//Chi*******************************************************
double Chi(int skolko)
{
	double np;
	int pow = s1 - n1 - k1;
	if (pow<0) pow = -pow;
	np = skolko*Power(2, pow);
	return (nu - np)*(nu - np) / (1.0*np);
}
//CalcChi
void CalcChi(int outs) {
	x2 = Chi(outs); sumx2 = sumx2 + x2;
	//printf("%u %.2f, ",nu,x2);
	if (x2>3.84) u05 = u05 + 1; if (x2>6.64) u01 = u01 + 1;
}
//PrintStats*****************************8

struct Stats {
	double perc01;
	double perc05;
	double avx2;
};

Stats getStats() {
	avx2 = sumx2 / keyquo; perc05 = 100.0*u05 / keyquo; perc01 = 100.0*u01 /
keyquo;
```

```
      sumx2 = 0; u05 = 0; u01 = 0;
      return{ perc01, perc05, avx2 };

}
void initstats() {
      u01 = 0; u05 = 0; sumx2 = 0;
}
//Hash*****************************
unsigned int nnnnn;
unsigned int hsh(unsigned int n)
{
      nnnnn = n;
      nnnnn ^= nnnnn >> 16;
      nnnnn *= 0x85ebca6b;
      nnnnn ^= nnnnn >> 13;
      nnnnn *= 0xc2b2ae35;
      nnnnn ^= nnnnn >> 16;
      return nnnnn%K1;
}
//Ins**********************************
void Ins(unsigned int x, LINK& p){
      if (p == NULL) {
            p = new Element(x);
            p->next = first;
            first->prev = p;
            first = p;
      }
      else Ins(x, p->right);
}
void Insert(unsigned int x) { Ins(x, ht[hsh(x)]); }
//Init*****************************
void Init(int kk, int nn, int ss)
{
      unsigned int i; unsigned int t;
      k1 = kk; n1 = nn; s1 = ss;
      N1 = Power(2, n1);
      K1 = Power(2, k1);
      ht = new LINK[K1];
      for (i = 0; i<K1; i++) ht[i] = NULL;
      t = 0;
      ht[hsh(t)] = new Element(t);
      last = ht[hsh(t)]; first = last;
      for (i = 1; i<K1; i++) Insert(i);
      //printf("HASH TABLE INITED K=%u\n",K1);
}
//Size**************************
unsigned int Size() {
      unsigned int i, vol = 0;
      LINK q;
      for (i = 0; i<K1; i++) {
            q = ht[i];
            while (q != NULL) { vol++; q = q->right; }
      }
      return vol;
}
//MaxLen***************************
unsigned int MaxLen() {
      unsigned int i, ml, max = 0;
      LINK q;
      for (i = 0; i<K1; i++) {
            q = ht[i]; ml = 0;
            while (q != NULL) { ml++; q = q->right; }if (max<ml) max = ml;
      }
```

```
      return max;
}
//AvLen***************************
double AvLen() {
      unsigned int i, ml, max = 0; unsigned int sum = 0;
      LINK q;
      for (i = 0; i<K1; i++) {
            q = ht[i]; ml = 0;
            while (q != NULL) { ml++; q = q->right; sum = sum + ml; }
      }
      return 1.0*sum / K1;
}
void Print() {
unsigned int i;
LINK q;
for (i=0;i<K1;i++) {
q=ht[i];printf("#%i->",i);
while (q!=NULL) {printf("%u %u,",q->value.int1,q->value.int2);q=q->right;}
printf("\n");}}
void PrintFL() {
LINK p=last;
while (p!=NULL)   {printf("%u %u, ",p->value.int1,p->value.int2);p=p->prev;}
printf("\n");}
void PrintFF() {
LINK p=first;
while (p!=NULL)   {printf("%u %u, ",p->value.int1,p->value.int2);p=p->next;}
printf("\n");}


void MoveToFront(LINK p) {
      if (p != first) {
            if (p == last) { last = last->prev; last->next = NULL; }
            else  { p->prev->next = p->next; p->next->prev = p->prev; }
            p->next = first; first->prev = p; p->prev = NULL; first = p;
      }
}

void SeaBS(unsigned int x, LINK& p){
      if (p == NULL) {
            if (last != ht[hsh(last->value)]) printf("!");
            if ((hsh(x) != hsh(last->value)) || (last->right != NULL))
            {
                  ht[hsh(last->value)] = last->right; p = last;
            }
            last->value = x;
            last->right = NULL;
            MoveToFront(last);
      }
      else {
            if (p->value == x) {
                  MoveToFront(p); is = 1;
                  if (p->right != NULL) {
                        LINK tem = p; p = p->right; LINK q = p;
                        while (q->right != NULL) q = q->right;
                        q->right = tem;
                        tem->right = NULL;
                  }
            }
            else SeaBS(x, p->right);
      }
}
void Search(unsigned int x) { is = 0; SeaBS(x, ht[hsh(x)]); }
```

ПРИЛОЖЕНИЕ Д

Программный код теста «адаптивный критерий хи-квадрат»

```
#include "stdafx.h"
#include "blockciphers.h"
#include "BookStack32.h"
#include <iostream>
#include <string>
#include <ctime>
#include <hash_set>
using namespace std;
int main(int argc, char **argv)
{

	srand(time(0));
	int rnds = 0;
	u32 blk[4];
	float T;
	srand(time(NULL));
	setCipher(AES);
	hash_set <unsigned long > hs1;//объявление хэш таблиц
	hash_set <unsigned long > ::const_iterator hs1_AcIter, hs1_RcIter;
//объявление переменных
	float t1 = clock();
	int k = 0;

	for (int i = 0; i < 1048576; i++) {
		{
			blk[0] = 0;
			blk[1] = 0;
			blk[2] = 0;
			blk[3] = i;
		}

		encrypt(blk, rnds);

		hs1.insert(blk[0]); //создание обучающей выборки, добавление в хэш
таблицу.
	}

	int temp;
	for (int i = 1048576; i < 17825792; i++) {
		{
			blk[0] = 0;
			blk[1] = 0;
			blk[2] = 0;
			blk[3] = i;
		}

		encrypt(blk, rnds);

		hs1_RcIter = hs1.find(blk[0]); //поиск элемента в хэш таблице и
вычисление количества найденных элементов
		if (hs1_RcIter == hs1.end())//если элемента нет
			k += 0;//тогда ничего и не прибавляем
		else
			//     cout<<p1[0]<<endl;
			k++;//иначе прибавляем +1
```

```
        }
        float t2 = clock();
        T = t2 - t1;
        cout << "time to test: " << T / 1000 << " seconds" << endl;

        system("pause");
        return 0;
}
```

ПРИЛОЖЕНИЕ Е

Программный код скрипта для генерации отчётов о тестировании NIST

```
from pathlib import Path
import subprocess
from matplotlib import pyplot
result_types = ('SUCCESS', 'FAIL')
def make_reports() :
result = list(Path("./ciphertexts/").rglob("*.bin"))
for filepath in result :
reports_folder = filepath.parent.joinpath('reports')
if not reports_folder.exists() :
reports_folder.mkdir()
input_file = str(filepath)
output_file = reports_folder.joinpath(filepath.name.replace('.bin', '.txt'))
print('start ', str(filepath))
args = ("./statisticaltests", input file, output file)
p = subprocess.Popen(args, stdout = subprocess.PIPE)
p.wait()
short_report_file = reports_folder.joinpath(filepath.name.replace('.bin',
'_short.txt'))
with output_file.open('r') as report, short_report_file.open('w') as
short_report :
for line in report.readlines() :
line = str(line)
if any(substr in line for substr in result_types) :
short_report.write(line)

print(str(p.stdout.read()))
print()

if __name__ == "__main__":
make_reports()
```

ПРИЛОЖЕНИЕ Ж

Программный код скрипта для генерации отчётов по проведенным тестированиям

```
from pathlib import Path
import subprocess
from matplotlib import pyplot
def make_graphics(pathname) :
cyphertext_path = Path(pathname)
for tmp in cyphertext_path.iterdir() :
if not tmp.is_dir() :
continue

reports = sorted(tmp.rglob("*_short.txt"))
# reports.sort()

number_of_success_tests = []
      for filename in reports :
      success_tests = 0
            with filename.open('r') as short_report :
      for line in short_report.readlines() :
      if "SUCCESS" in line and "FAILURE" not in line :
      success_tests += 1
            number_of_success_tests.append(success_tests)

            title = str(tmp)
            title = title[title.rfind('/') + 1:]
            pyplot.title(title)

            ticks = list(range(3, len(number of success_tests) + 3, 3))
            pyplot.xticks(ticks, labels = ticks)
            pyplot.xlabel('Количество раундов')
            pyplot.ylabel('Количество пройденных тестов')

            pyplot.plot(range(3, len(number_of_success_tests) + 3),
number_of_success_tests)
            pyplot.yticks(range(0, 140, 5), labels = range(0, 140, 5))
            pyplot.savefig(tmp.joinpath(f"reports/graph_{title}.png"))
            pyplot.close()
            print(tmp)
            print(reports)
            print(number_of_success_tests)
            print()

            def make_report_names_sorted() :
            reports = list(Path("./ciphertexts").rglob('*_short.txt'))
      for report in reports :
      short_report_name = report.name
      if short_report_name[0].isdigit() :
            continue
            short_report_name = short_report_name[short_report_name.find('_')
+ 1:]

            if len(short_report_name[:short_report_name.find('_')]) == 1 :
                  short_report_name = '0' + short_report_name
                  short_name = report.parent.joinpath(short_report_name)
                  report.rename(short_name)
            if __name__ == "__main__" :
                  make_report_names_sorted()
                  make_graphics("./ciphertexts/uniblocks")
                  make_graphics("./ciphertexts/cppcrypto")
```

ПРИЛОЖЕНИЕ 3

Заголовочный файл Uniblocks

```
#include "stdafx.h"
#include "Ciphers/BookStack32.h"
#include "utils.h"
#include <iostream>
#include <string>
#include <ctime>
#include <vector>
#include <fstream>
#include "blockciphers.h"

using namespace std;

vector<unsigned char> intToBytes(u64 paramInt)
{
	vector<unsigned char> arrayOfByte(8);
	for (int i = 0; i < 8; i++)
		arrayOfByte[7 - i] = (paramInt >> (i * 8));
	return arrayOfByte;
}

u64 nonce = 0x1F3C091AD5B3CFAB;
u64 counter = 0x0000000000000000;
//функции для генерации текстов в режиме ctr
void generate_text(u32* result) {
	u16 nonce16[4];
	u16 counter16[4];
	_64to16(&nonce, nonce16, 1);
	_64to16(&counter, counter16, 1);

	for (int i = 0; i < 4; i++) {
		u16 temp[2] = { counter16[3 - i], nonce16[3 - i] };
		_16to32(temp, &(result[i]), 1);
	}
}

u64 next_number(int rounds) {
	u32 text32[4];
	generate_text(text32);

	encrypt(text32, rounds);

	u64 result;
	_32to64(&result, text32, 1);
	result ^= 0;
	if (counter + 1 == INT64_MAX) {
		counter = 0;
	}
	else {
		counter++;
	}

	return result;
}

void cipherTestGeneration();

int main(int argc, char **argv)
{
	cipherTestGeneration();
```

```
}
void cipherTestGeneration() {
	srand(time(0));
	u32 blk[4];
	//Test
	unsigned int i;
	keyquo = 1; //Количество случайных ключей, для которых осуществляется
проверка
	int kol = 0;
	FILE* rfile = fopen("resultlk24 (s=32, 30).txt", "w");
	FILE* rsfile = fopen("resultlk24 (s=32, 30)_sum.txt", "w");
	Init(18, 24, 32);
	fprintf(rfile, "k=%u n=%u s=%u keyquo=%d\n", k1, n1, s1, keyquo);
	printf("k=%u n=%u s=%u keyquo=%d\n", k1, n1, s1, keyquo);

	fprintf(rfile, "%10s %10s %10s %10s\n", "Cipher", "perc05", "perc01",
"avx2");
	printf("%10s %10s %10s %10s\n", "Cipher", "perc05", "perc01", "avx2");

	for (int cip = LBLOCK; cip <= KTANTAN64; cip++) {
		setCipher((Ciphers)cip);
		fprintf(rsfile, "%c%c%c ", id()[0], id()[1], id()[2]);
	}
	fprintf(rsfile, "\n");

	int rnds = 2;
	do {

		//Инициализация теста 2^{18} - размер верхней части стопки книг,
		//2^{18} - размер тестируемых выборок
		//2^{32} - размер алфавита, т.е. 32 - размер тестируемых слов
		kol = 0;
		rnds++;
		fprintf(rfile, "ROUNDS=%d\n", rnds);
		printf("ROUNDS=%d\n", rnds);

		for (int cip = LBLOCK; cip <= KTANTAN64; cip++) {
			setCipher((Ciphers)cip);
			fprintf(rfile, "%10s: ", id());
			printf("%10s: ", id());

			counter = 0;
			string filename = "ciphertexts/uniblocks/" + (string)id() +
"/" + (string)id() + "_" + to_string(rnds) + "_ciphertext.bin";
			cout << filename << endl;
			ofstream fout;
			fout.open(filename, ios::binary | ios::out);
			//Start
			for (keyn = 0; keyn < keyquo; keyn++) {
				resetKey();
				//записываем 16 мб текста прогоняя шифры в режиме ctr
				for (int j = 0; j < 2097152; ++j) { // 16 MB
					//вызов
					vector<unsigned char> bytes =
intToBytes(next_number(rnds));

					for (unsigned char &byte : bytes) {
						fout.write((char *)&byte, sizeof(byte));
					}
				}
			}
			fout.close();
			Stats st = getStats();
```

```
                  fprintf(rfile, "%10.0f %10.0f %10.2f", st.perc05, st.perc01,
st.avx2);
                  printf("%10.0f %10.0f %10.2f", st.perc05, st.perc01,
st.avx2);

                  if (st.perc01 <= 10 && st.perc05 <= 20) {
                        fprintf(rfile, " --- NO\n");
                        printf(" --- NO\n");
                        fprintf(rsfile, "%3s ", "NO");
                  }
                  else {
                        fprintf(rfile, " +++ YES\n");
                        printf(" +++ YES\n");
                        fprintf(rsfile, "%3s ", "YES");
                        kol++;
                  }

                  //вывод статистики по всем выборкам

            }
            fprintf(rsfile, "\n");
            fprintf(rfile, "KOL=%d\n", kol);
            printf("KOL=%d\n", kol);
            fprintf(rfile, "-----------------------------------------------\n",
kol);
            printf("-----------------------------------------------\n", kol);

      } while (rnds <= 30);

      fclose(rfile);
      fclose(rsfile);

      system("pause");
}
```

ПРИЛОЖЕНИЕ И

Код утилиты TxtToIMG Utility

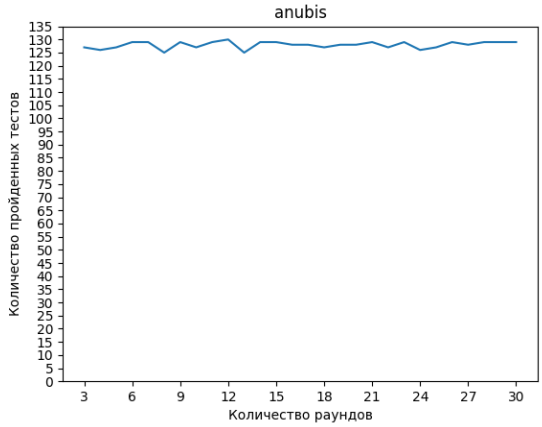
```
#include "stdafx.h"
#include <string>;
#include <Windows.h>
#include <tchar.h>
#include <time.h>
#include <iostream>
#include <gdiplus.h>
using namespace std;
using namespace Gdiplus;
#pragma comment (lib, "Gdiplus.lib")

int GetEncoderClsid(const WCHAR* format, CLSID* pClsid)
{
   UINT  num = 0;          // number of image encoders
   UINT  size = 0;         // size of the image encoder array in bytes

   ImageCodecInfo* pImageCodecInfo = NULL;

   GetImageEncodersSize(&num, &size);
   if(size == 0)
      return -1;  // Failure

   pImageCodecInfo = (ImageCodecInfo*)(malloc(size));
   if(pImageCodecInfo == NULL)
      return -1;  // Failure

   GetImageEncoders(num, size, pImageCodecInfo);

   for(UINT j = 0; j < num; ++j)
   {
      if( wcscmp(pImageCodecInfo[j].MimeType, format) == 0 )
      {
         *pClsid = pImageCodecInfo[j].Clsid;
         free(pImageCodecInfo);
         return j;  // Success
      }
   }

   free(pImageCodecInfo);
   return -1;  // Failure
}

VOID Example_SaveFile(const WCHAR *F, const WCHAR *T)
{
      GdiplusStartupInput gdiplusStartupInput;
      ULONG_PTR gdiplusToken;
      GdiplusStartup(&gdiplusToken, &gdiplusStartupInput, NULL);

      CLSID   encoderClsid;
      Status  stat;
      Image*   image = new Image(F);

                                                  // Get the CLSID of the PNG
encoder.
      GetEncoderClsid(L"image/jpeg", &encoderClsid);

      stat = image->Save(T, &encoderClsid, NULL);

      if (stat == Ok)
            printf("img was saved successfully\n");
```

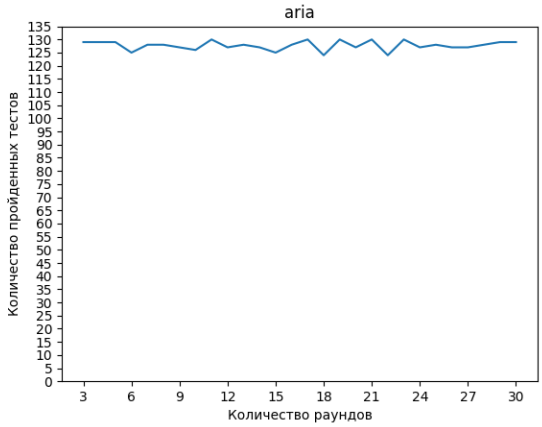
```
	else
		printf("Failure: stat = %d\n", stat);

	delete image;
	GdiplusShutdown(gdiplusToken);

}

#define  DIB_RGB(r, g, b) \
((DWORD)((r & 0xFF) << 16) | ((g & 0xFF) <<                                    8)
| (b & 0xFF))


void draw(__int32* arr, int width, int x, int y, DWORD color) {
	for (int r = y; r <= y; r++) {
		for (int c = x; c <= x; c++)
			arr[r*width + c] = color;
	}
}
// 24/32 бит
BOOL  SaveArrFile(const TCHAR* filename, const __int32* arr,
	int width, int height, int bpp = 24) {

	if ((bpp < 24) || (bpp > 32)) // только 24/32 бит
		return FALSE;

	DWORD p_row = (DWORD)((width * bpp + 31) & ~31) / 8uL;
	DWORD size = (DWORD)(height * p_row);

	// формируем файловый заголовок
	BITMAPFILEHEADER  hdr;
	ZeroMemory(&hdr, sizeof(BITMAPFILEHEADER));
	hdr.bfType = 0x4D42;
	hdr.bfOffBits = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER);
	hdr.bfSize = hdr.bfOffBits + size;

	// заголовок описателя растра
	BITMAPINFO dib;
	ZeroMemory(&dib, sizeof(BITMAPINFO));
	dib.bmiHeader.biSize = sizeof(BITMAPINFOHEADER);
	dib.bmiHeader.biBitCount = bpp;
	dib.bmiHeader.biCompression = BI_RGB;
	dib.bmiHeader.biPlanes = 1u;
	dib.bmiHeader.biWidth = (long)width;
	dib.bmiHeader.biHeight = (long)-height;
	dib.bmiHeader.biSizeImage = size;
	dib.bmiHeader.biXPelsPerMeter = 11811L;
	dib.bmiHeader.biYPelsPerMeter = 11811L;
	dib.bmiHeader.biClrImportant = 0uL;
	dib.bmiHeader.biClrUsed = 0uL;

	// далее запись в файл
	HANDLE fp = CreateFile(filename, GENERIC_WRITE, FILE_SHARE_WRITE, NULL,
		CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL);
	if (fp == INVALID_HANDLE_VALUE)
		return FALSE;

	// записываем заголовки...
	DWORD  dwr = 0uL;
	WriteFile(fp, (LPCVOID)&hdr, sizeof(BITMAPFILEHEADER), &dwr, NULL);
	WriteFile(fp, (LPCVOID)&dib.bmiHeader, sizeof(BITMAPINFOHEADER), &dwr,
NULL);
```

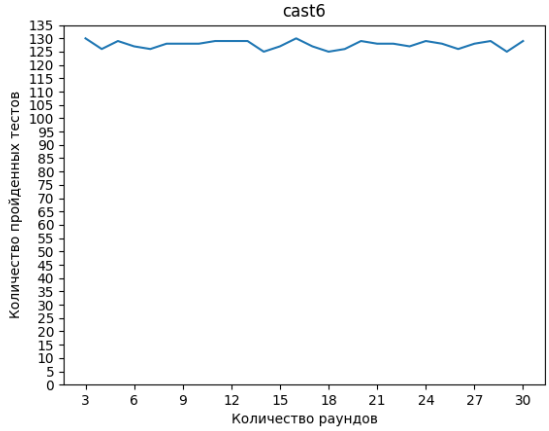
```
	// запись массива пикселей
	if (bpp == 32) // 32-бит
		WriteFile(fp, (LPCVOID)arr, size, &dwr, NULL);
	else if (bpp == 24) { // 24-бит с дополнением до 32-разрядной границы

		BYTE   nil = 0u;
		int   cb = sizeof(RGBQUAD);
		int  align = ((cb - ((width*bpp + 7) / 8) % cb) % cb);

		for (int y = 0; y < height; y++) {
			for (int x = 0; x < width; x++)
				WriteFile(fp, (LPCVOID)&arr[y*width + x],
sizeof(RGBTRIPLE), &dwr, NULL);

			for (int i = 0; i < align; i++) // до границы DWORD
				WriteFile(fp, (LPCVOID)&nil, sizeof(BYTE), &dwr,
NULL);
		}
	}

	FlushFileBuffers(fp);
	CloseHandle(fp);
	return TRUE;
}




int  main(void) {
	string fnm;
	srand(time(NULL));
	//массив пикселей
	__int32 arr[400 * 400] = { 0 };
	int cw = 400;
	int ch = 400;
	int counter = 0;

	char s1, s2, s3;

	// нарисуем что-нибудь
	DWORD rgb;
	//FILE *ptrFile; fopen_s(&ptrFile, "ciphertext.txt", "rb");
	char label[] = "simon_";
	char FileName[256];

			for (int i = 1; ; i++) {
				char folder1[]= "input\\Simon_1\\";

				char folder2[]= "output\\Simon_1\\";

				//CreateDirectory(folder2, NULL);
				sprintf(FileName, "%s%s%i%s", folder1, label, i,
".txt");
				try {
					FILE *ptrFile;
					if (fopen_s(&ptrFile, FileName, "rb"))
						throw 1;
					unsigned char Red, Green, Blue;
					for (int y = 0; y < ch; y++) {
						for (int x = 0; x < cw; x++) {
							Red = fgetc(ptrFile);
```

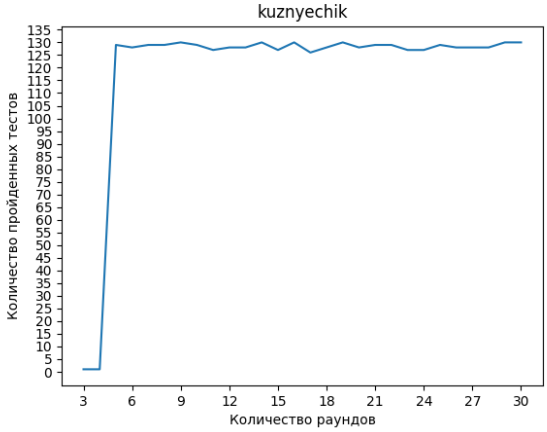
```
                                    //     cout<<(int)Red<<" ";
                                    Green = fgetc(ptrFile);
                                    //           cout<< (int)Green<<" ";
                                    Blue = fgetc(ptrFile);
                                    //            cout<< (int)Blue<<" ";

                                    rgb = DIB_RGB(int(Red), int(Green),
int(Blue));
                                    draw(arr, cw, x, y, rgb);

                              }

                        }

                        char cipher[] = "simon_";
                        char OutputFile[256];
                        // сохраняем в файл
                        sprintf(OutputFile, "%s%i%s", cipher, i,
".bmp");
                        fnm = cipher;
                        fnm += to_string(i);
                        if (SaveArrFile(_T(OutputFile), arr, cw, ch,
24))
                              cout << "Excellent save file N - " << i <<
endl;
                        else
                              _putts(_T("Error save file !"));

                        string F1 = fnm + ".bmp";
                        string F2 = folder2 + fnm + ".jpg";
                        wstring F11 = wstring(F1.begin(), F1.end());
                        wstring F21 = wstring(F2.begin(), F2.end());
                        const WCHAR *FNMBM = F11.c_str();
                        const WCHAR *FNMJPG = F21.c_str();
                        Example_SaveFile(FNMBM, FNMJPG);
                        if (remove(F1.c_str())) {
                              printf("Error removing file\n");
                        }


                  }
                  catch (...) {
                        break;
                  }


      }
```

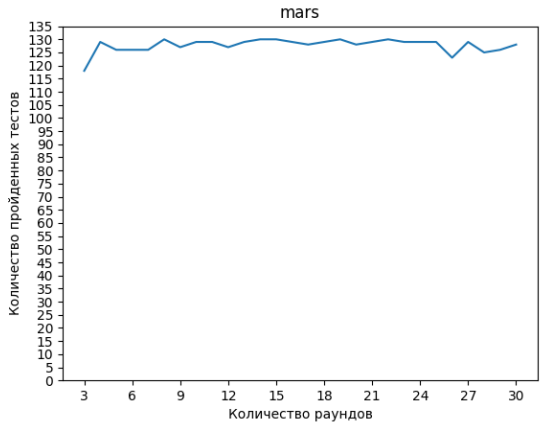
ПРИЛОЖЕНИЕ К

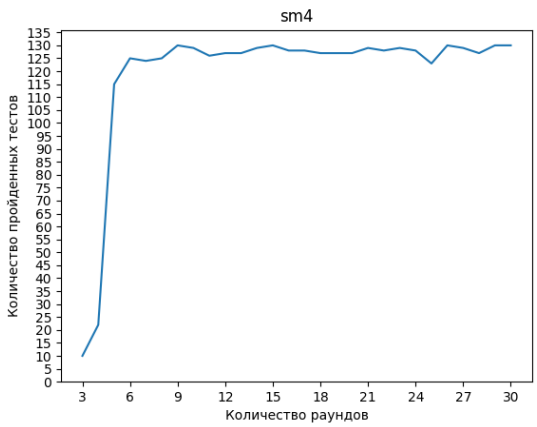


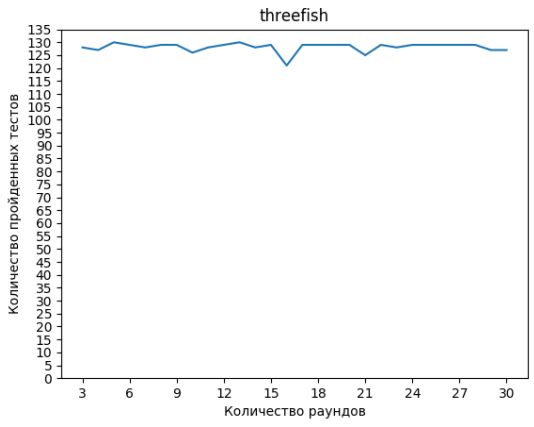


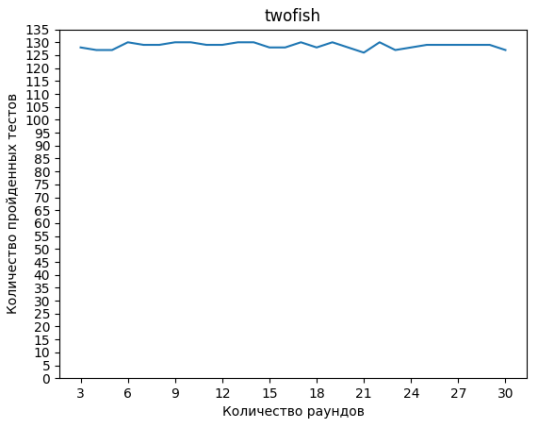


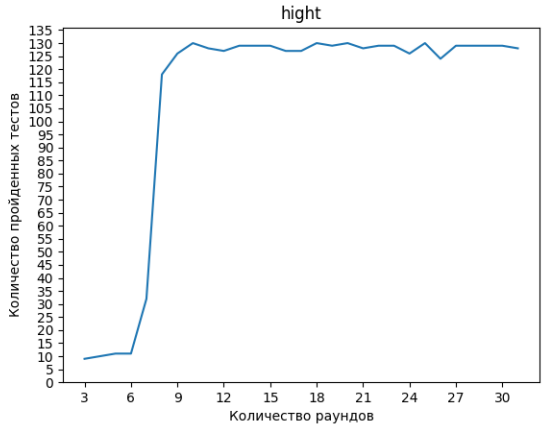


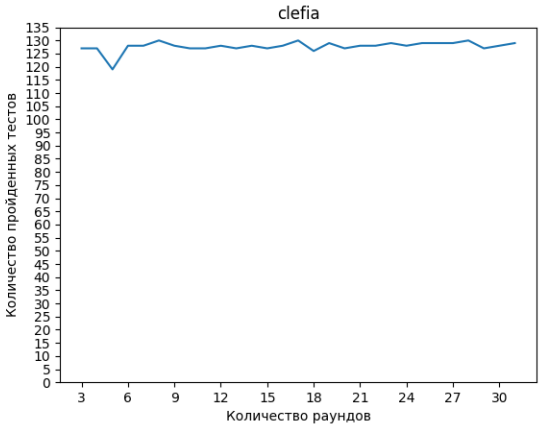


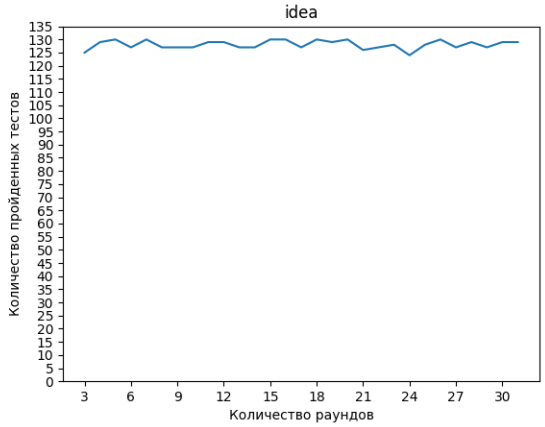


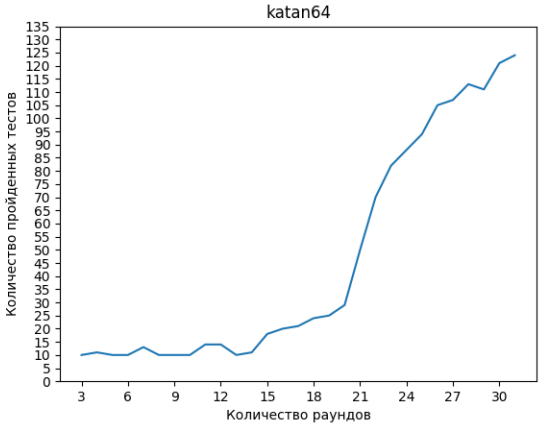


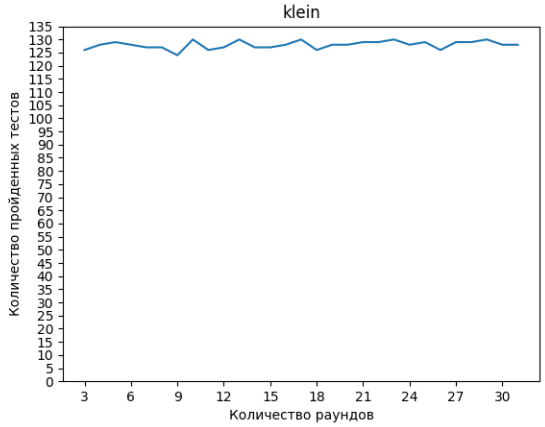


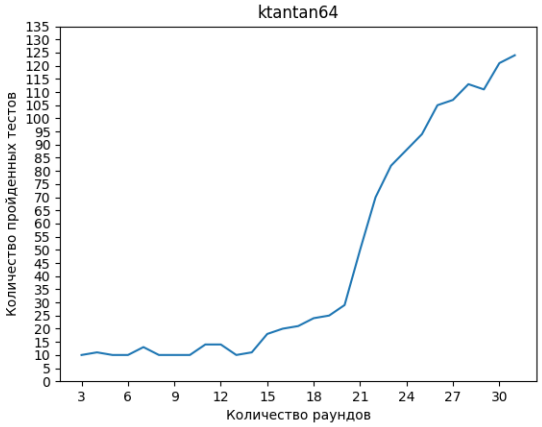

hight
Количество пройденных тестов
Количество раундов

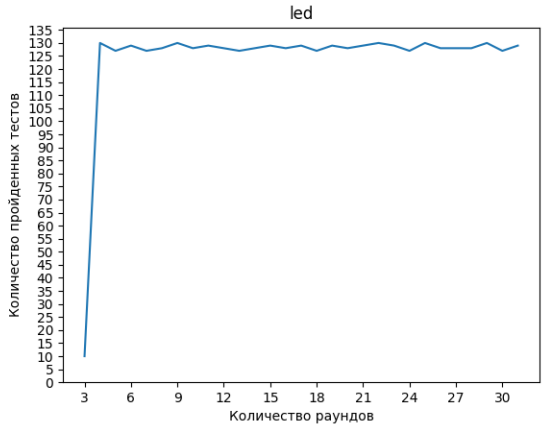

clefia
Количество пройденных тестов
Количество раундов

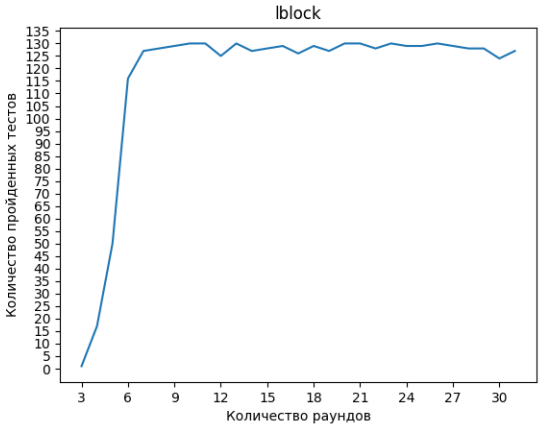

idea
Количество пройденных тестов
Количество раундов

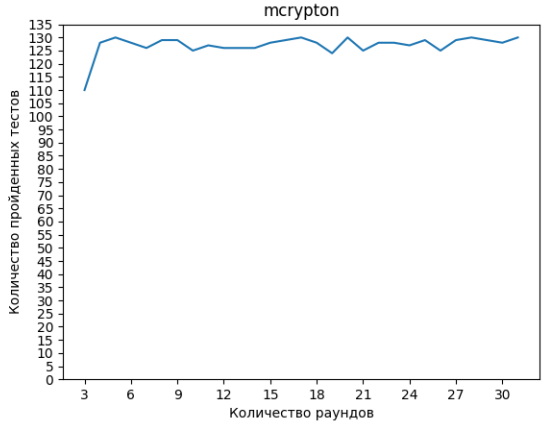

katan64
Количество пройденных тестов
Количество раундов

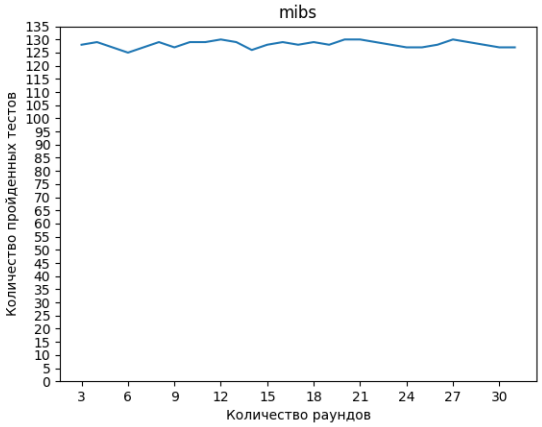

klein
Количество пройденных тестов
Количество раундов

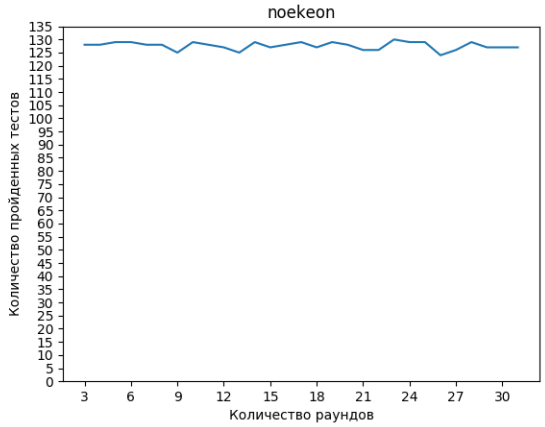

ktantan64
Количество пройденных тестов
Количество раундов

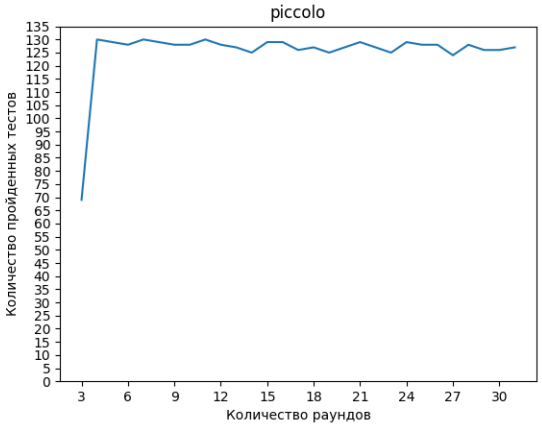

led
Количество пройденных тестов
Количество раундов

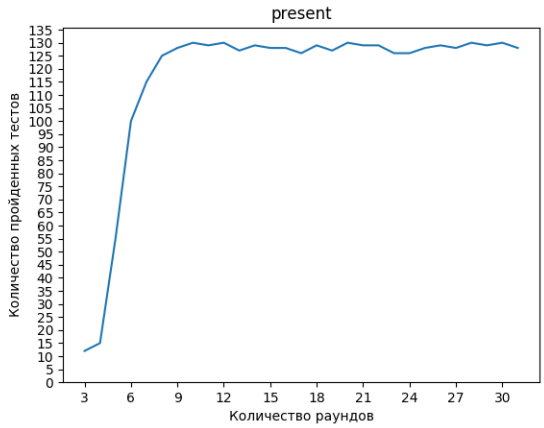

lblock
Количество пройденных тестов
Количество раундов

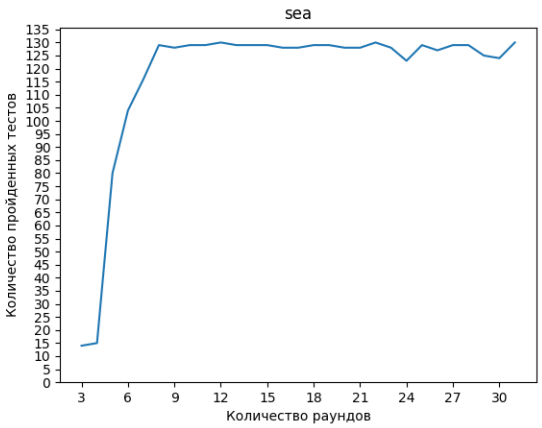

mcrypton
Количество пройденных тестов
Количество раундов

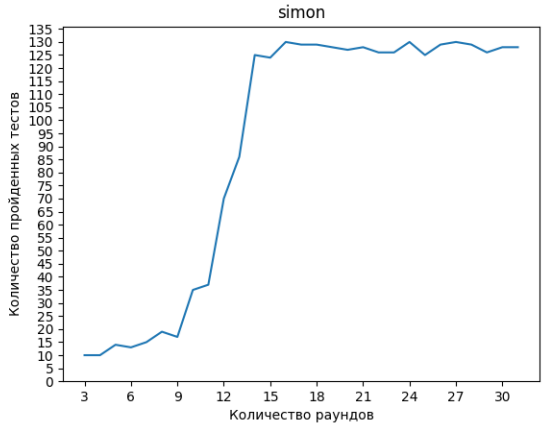

mibs
Количество пройденных тестов
Количество раундов

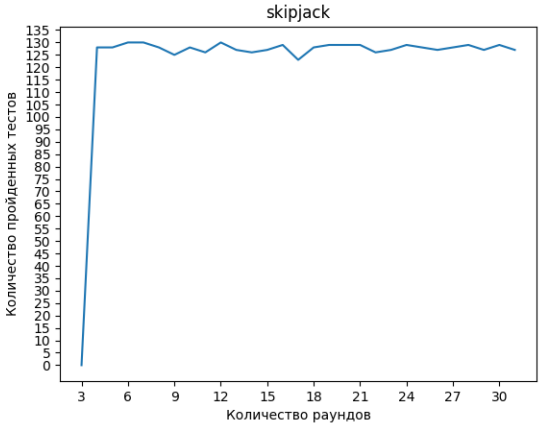

noekeon
Количество пройденных тестов
Количество раундов

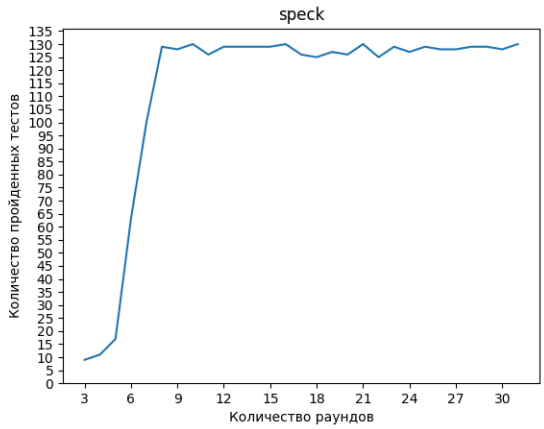

piccolo
Количество пройденных тестов
Количество раундов

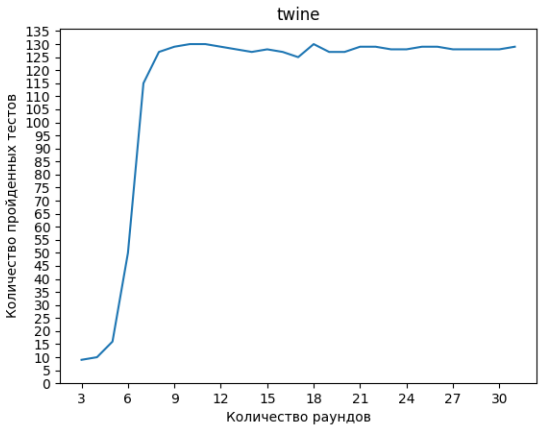

present
Количество пройденных тестов
Количество раундов

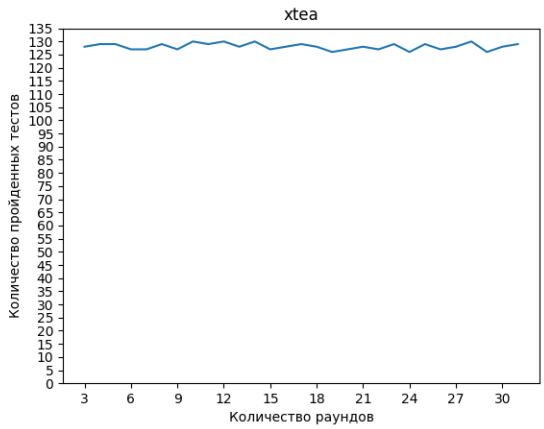

sea
Количество пройденных тестов
Количество раундов


simon
Количество пройденных тестов
Количество раундов


skipjack
Количество пройденных тестов
Количество раундов


speck
Количество пройденных тестов
Количество раундов


twine
Количество пройденных тестов
Количество раундов


xtea
Количество пройденных тестов
Количество раундов

ПРИЛОЖЕНИЕ Л

Научно-исследовательский институт
информационно-коммуникационных технологий

ООО «НИИ ИКТ» ИНН 5406600336, КПП 540601001, ОГРН 1165476054567
630099, РФ, г. Новосибирск, ул. Романова, д. 60, 232, т/ф +79132085555, office@nii-ikt.ru
Р/С № 40702 810 8231 3000 2113 филиал «Новосибирский» АО «АЛЬФА-БАНК»,
БИК 045004774, К/С 30101810600000000774

СПРАВКА О ВНЕДРЕНИИ

результатов диссертационной работы Перова Артема Андреевича на тему «Универсальный метод построения решающих правил с использованием сверточных нейронных сетей для анализа генераторов псевдослучайных последовательностей на основе итеративных блочных шифров», представляемой на соискание ученой степени кандидата технических наук по специальности 05.13.17 – «Теоретические основы информатики»

Настоящей справкой подтверждается, что результаты диссертационной работы А.А. Перова используются в практической деятельности Научно-исследовательского института информационно-коммуникационных технологий. Программно-аппаратный комплекс, разработанный в рамках диссертационного исследования, применяется для оценки статистических свойств псевдослучайных последовательностей, полученных посредством генераторов на основе итеративных блочных шифров.

Применение данного программного комплекса позволяет подобрать такое количество раундов генератора, которое обеспечивает удовлетворительные статистические свойства псевдослучайных последовательностей при приемлемой производительности.

«24» ноября 2020 г. _____ / к.т.н., директор НИИ ИКТ
Басыня Евгений Александрович

ПРИЛОЖЕНИЕ М

АКТ О ВНЕДРЕНИИ

результатов диссертационной работы Перова Артёма Андреевича «Универсальный метод построения решающих правил с использованием сверточных нейронных сетей для анализа генераторов псевдослучайных последовательностей на основе итеративных блочных шифров»

Настоящим актом удостоверяется, что результаты диссертационного исследования Перова Артёма Андреевича на тему: «Универсальный метод построения решающих правил с использованием сверточных нейронных сетей для анализа генераторов псевдослучайных последовательностей на основе итеративных блочных шифров» были использованы ООО «Акстел-Безопасность» в задачах исследования эффективности работы криптографических средств защиты информации, в частности, итеративных блочных шифров.

Внедрение предлагаемого в диссертационной работе метода позволило:

- провести статистический анализ используемых в программно-аппаратных средствах защиты, реализуемых компанией, блочных шифрах;
- использовать усеченные значения характеристик работы шифров, что в некоторых случаях существенно ускорило процесс криптографического преобразования.

Результаты исследования, изложенные в диссертации, имеют важное практическое значение. Проведенные экспериментальные работы продемонстрировали высокий уровень достоверности результатов.

Генеральный директор
ООО «Акстел-Безопасность»
_____ Трабакин С.В. 14.09.2020