Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ”

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

ЭВОЛЮЦИОННЫЕ АЛГОРИМТЫ ДЛЯ КРИПТОАНАЛИЗА ГЕНЕРАТОРОВ КЛЮЧЕВОГО ПОТОКА С ИСПОЛЬЗОВАНИЕММ ПРОГРАММНЫХ СРЕДСТВ РЕШЕНИЯ ЗАДАЧИ ВЫПОЛНИМОСТИ

Автор *Павленко Артём Леонидович* _____

Направление подготовки *01.03.02 Прикладная математика и информатика*

Квалификация *Бакалавр*

Руководитель *Ульянцев В. И., к.т.н.* _____

К защите допустить

Зав. кафедрой КТ *Васильев В.Н., проф., д.т.н.* _____

“___” _____ 2018 г.

Санкт-Петербург, 2018 г.

Студент _____*Павленко А.Л.*_____ Группа _*М3437*_ Кафедра _*КТ*_ Факультет _*ИТиП*_

Направленность (профиль), специализация
Математические модели и алгоритмы в разработке программного обеспечения

ВКР принята «____» _____ 2018 г.

Оригинальность ВКР _____%

ВКР выполнена с оценкой _____

Дата защиты «____» июня 2018 г.

Секретарь ГЭК *Павлова О.Н.*_____ _____

Листов хранения _____

Демонстрационных материалов/Чертежей хранения _____*отсутствуют*_____

Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ”

УТВЕРЖДАЮ

Зав. кафедрой *КТ*

проф. Васильев В. Н

_____ 2018 г.

З А Д А Н И Е
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ

Студенту *Павленко А.Л.* **Группа** *М3437* **Кафедра** *КТ* **Факультет** *ИТиП*

Руководитель *Ульянцев В.И., к.т.н., доцент кафедры КТ*

1 Наименование темы: *Эволюционные алгоритмы для криптоанализа генераторов ключевого потока с использованием программных средств решения задачи выполнимости*

Направление подготовки (специальность) *01.03.02 Прикладная математика и информатика*

Квалификация *Бакалавр*

2. Срок сдачи студентом законченной работы *15 мая 2018 г.*

3. Техническое задание и исходные данные к диссертации

Требуется разработать и реализовать эволюционный алгоритм для автоматизированного построения декомпозиционных представлений трудных вариантов задач о булевой выполнимости и применить эти методы для построения guess-and-determine атак на ряд криптографических генераторов ключевого потока. В качестве исходных данных будут использованы наработки и результаты, полученные в лаборатории дискретного анализа и прикладной логики ИДСТУ СО РАН.

4 Содержание выпускной работы (перечень подлежащих разработке вопросов)

1. Постановка задачи. Обзор используемых программных средств. Обзор исследуемых криптографических алгоритмов. Обзор существующих автоматических методов для построения guess-and-determine атак.

2. Описание процесса разработки и реализации эволюционного алгоритма. Подбор параметров для повышения качества получаемых результатов и описание особенностей разрабатываемой программной среды. Описание нового метода адаптивного изменения объема выборки по ходу работы алгоритма.

3. Проведение экспериментов с различными генераторами ключевого потока. Сравнение результатов, полученных в ходе исследований, и результатов, описанных в соответствующих статьях.

5 Перечень графического материала (с указанием обязательного материала)

Графические материалы и чертежи работой не предусмотрены

6 Исходные материалы и пособия

1. *Semenov A., Zaikin O. Algorithm for Finding Partitionings of Hard Variants of Boolean Satisfiability Problem with Application to Inversion of Some Cryptographic Functions // SpringerPlus. – 2016. –Vol. 5. –P. 554-554*
2. *Semenov A., Zaikin O. Otpuschennikov I., Kochemazov S., Ignatiev A. On cryptographic attacks using backdoors for SAT // The Thirty-Second AAAI Conference on Artificial Intelligence, – IEEE, 2018. – P. 6641-6648*
3. *Metropolis N., Ulam S. The Monte Carlo Method // Journal of the American Statistical Association. – 1949. – Vol. 44. – No. 247. P. 335-341*

7 Дата выдачи задания «21» «ноября» 2017г.

Руководитель _____

Задание принял к исполнению_____

«21» «ноября» 2017г.

Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

“САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ”

АННОТАЦИЯ

ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

Студента *Павленко Артём Леонидович*

Наименование темы ВКР: *Эволюционные алгоритмы для криптоанализа генераторов ключевого потока с использованием программных средств решения задачи выполнимости*

Наименование организации, где выполнена ВКР *Университет ИТМО*

ХАРАКТЕРИСТИКА ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЫ

1 Цель исследования *Разработка и реализация автоматизированных методов построения декомпозиционных представлений трудных вариантов задач о булевой выполнимости*

2 Задачи, решаемые в ВКР *Разработка алгоритма для автоматизированного построения guess-and-determine атак на генераторы ключевого потока. Сравнение результатов, полученных в ходе исследований, и результатов, описанных в соответствующих статьях*

3 Число источников, использованных при составлении обзора *17*

4 Полное число источников, использованных в работе *17*

5 В том числе источников по годам

Отечественных			**Иностранных**		
Последние 5 лет	От 5 до 10 лет	Более 10 лет	Последние 5 лет	От 5 до 10 лет	Более 10 лет
0	*2*	*0*	*4*	*10*	*1*

6 Использование информационных ресурсов Internet *да, 7*

7 Использование современных пакетов компьютерных программ и технологий

Пакеты компьютерных программ и технологий	**Параграф работы**
PyCharm	*2*
minisat	*1, 2*
Lingeling	*1, 2*
ROKK	*1, 2*

8 Краткая характеристика полученных результатов

Разработан алгоритм для автоматизированного построения guess-and-determine атак на генераторы ключевого потока. Предложен метод адаптивного изменения объема выборки по ходу работы алгоритма. Проведено сравнение работы алгоритма с использованием предложенного метода и без него. Произведены сравнения результатов, полученных в ходе исследований, и результатов, описанных в соответствующих статьях. Получено новое декомпозиционное множество для шифра Trivium 64.

9 Полученные гранты, при выполнении работы

При выполнении работы грантов получено не было.

10 Наличие публикаций и выступлений на конференциях по теме выпускной работы *да*

Публикации:

1. *Павленко А.Л., Ульянцев В.И. Эволюционные алгоритмы для криптоанализа генераторов ключевого потока с использованием программных средств решения задачи выполнимости // Сборник тезисов докладов конгресса молодых ученых. Электронное издание*

Конференции

2. *VII Конгресс молодых ученых 2018 (получен диплом за лучший научно-исследовательский доклад)*

Студент *Павленко А. Л.* _____

Руководитель *Ульянцев В. И.* _____

“_____” _____ 2018 г.

Оглавление

Введение

В рамках данной работы предполагается разработать и реализовать новые автоматизированные методы построения декомпозиционных представлений трудных вариантов задач о булевой выполнимости и применить эти методы для построения атак на ряд криптографических генераторов ключевого потока.

В настоящее время уровень безопасности систем и протоколов передачи данных критически зависит от стойкости криптографических алгоритмов, которые применяются для защиты данных в них. Автоматизированные методы, которые позволяют оценивать стойкость таких алгоритмов, являются ценными ввиду большого числа различных алгоритмов такого рода. Чтобы такие методы были эффективными, в их основе должна лежать комбинаторная задача с хорошо развитой алгоритмикой. Задача о булевой выполнимости, сокращенно обозначаемая *SAT*, хорошо зарекомендовала себя в этом плане.

Современные алгоритмы решения *SAT* успешно используются в криптоанализе, что позволяет рассматривать их в качестве основы для построения атак на многие известные криптографические примитивы и шифры. Актуальность настоящей работы обоснована тем, что построенные методы могут быть использованы для оценки уровня криптографической стойкости широкого спектра шифров. Представленные в работе алгоритмы могут использоваться в роли экспертных систем и оценки стойки различных криптографических функций.

Научная новизна работы состоит в том, что для поиска декомпозиционных представлений задач криптоанализа использовались эволюционные алгоритмы, тогда как в опубликованных ранее работах применялись метаэвристические схемы *поиск с запретами* и *имитация*

отжига. Также в настоящей работе была предложена новая эвристика *адаптивное изменение объема выборки*.

В практической части работы реализована программная среда для автоматизированного построения атак, основанных на алгоритмах решения *SAT*, на криптографические функции. Также были построены новые атаки из класса *guess-and-determine* для ряда современных шифров: *Trivium 64*, *A5/1*, *Bivium*.

Первая глава посвящена обзору используемых программных средств, таких как: автоматические трансляторы и *SAT*-решатели. Также в ней будут рассмотрены исследуемые криптографические алгоритмы и уже существующие автоматизированные методы для поиска декомпозиционных представлений трудных *SAT*-задач, ориентированные на построения атак на эти алгоритмы.

Во второй главе будут описаны этапы разработки и реализации автоматизированных методов, которые основываются на эволюционных алгоритмах. Будут предложены новые эвристики, которые позволяют уменьшить число производимых вычислений при поиске декомпозиционных множеств и повысить эффективность получаемых guess-and-determine атак.

В третьей главе будут приведены результаты экспериментов, которые были проведены в ходе выполнения работы. Будет проведено сравнение между эвристическими схемами *поиск с запретами* и *эволюционным алгоритмов*.

В заключении будет кратко приведены выводы по главам настоящей работы, а также описаны дальнейшие перспективы развития для данной темы.

Глава 1. Введение в предметную область и постановка задачи

В настоящее время любая система, в которой требуется защита данных, не может обойтись без использования криптографических алгоритмов. Особую ценность представляют алгоритмы, с высоким показателем стойкости к различным атакам. Но для хорошей оценки этого параметра, в большинстве случаев, требуется скрупулезный и нетривиальный анализ особенностей рассматриваемого алгоритма.

В данной работе будут рассмотрены автоматизированные методы, которые позволяют строить *guess-and-determine* атаки на различные генераторы ключевого потока.

1.1. Термины и определения

В данном разделе приводятся термины и определения, которые будут использоваться в других частях этой работы. Описание некоторых из них не претендует на полноту, но является достаточным и корректным для понимания данной работы.

1.1.1. Элементы теории сложности

SAT – алгоритмическая задача выполнимости булевых формул. Экземпляром такой задачи является булева формула состоящая из имен переменных, скобок и логических операций: *И* (конъюнкция), *ИЛИ* (дизъюнкция), *НЕ* (отрицание). Задача же заключается в том, можно ли найти такой набор значений логических переменных, чтобы удовлетворить заданную формулу. То есть присвоить всем переменным значения *истина* или *ложь*, чтобы сама формула приняла значение *истина*.

Конъюнктивная нормальная форма (КНФ или CNF) – нормальная форма, которая представлена в виде конъюнкции дизъюнкций литералов.

SAT-решатель (SAT-solver) – программное средство решения *SAT*-задачи. Позволяет строить такой набор значений переменных булевой формулы, что она обращается в *истину*, или доказательство, что такого набора не существует.

Unit Propagation – один из способов упрощения булевых формул. Если среди всего множества дизъюнктов существует такой что, он является литералом, то упрощение происходит по следующему правилу:

1. Если другой дизъюнкт содержит этот литерал, то этот дизъюнкт удаляется.
2. Если другой дизъюнкт содержит отрицание этого литерала, то литерал удаляется из этого дизъюнкта.

1.1.2. Метаэвристические алгоритмы

Поиск с запретами (Tabu search) – метаэвристический алгоритм локального поиска, отличительной особенностью которого является поддержание *списка запретов* (*tabu list*). Данный список включает в себя уже пройденные точки пространства поиска. В течение определенного времени возврат к этим решениям не возможен, что вынуждает алгоритм продолжать поиски новых решений.

Имитация отжига (Simulated annealing) – метаэвристический алгоритм локального поиска, который отличается от классического на этапе принятия решения о замене потенциального решения на новое. Если найденное решение лучше исходного, то замена делается всегда. Однако, в обратном случае, вероятность замены определяется функцией с некоторым параметром t (*temperature*), который медленно уменьшается до нуля по ходу выполнения алгоритма. Чем больше значение параметра t и меньше разница между двумя решениями, тем больше вероятность замены. Обычно алгоритм инициализируется с большим значением параметра t, что

вынуждает его случайно блуждать в пространстве значений исследуемой функции. Однако, с уменьшением *t* алгоритм все больше вырождается в классический алгоритм локального поиска.

Эволюционные алгоритмы (Evolutionary algorithms) – алгоритмы, основывающиеся на эволюционных вычислениях, которые, в свою очередь, заимствуют подходы популяционной биологии, генетики и эволюции. Традиционно подразделяются на *генетические алгоритмы* (*genetic algorithms*) и *эволюционные стратегии* (*evolution strategies*). Также их можно классифицировать на устойчивые, в которых на каждой итерации обновляется лишь часть найденных решений, и поколенческие – полное обновление решений на каждой итерации.

1.1.3. Термины криптоанализа

Секретный ключ (Secret key) – информация, используемая криптографическим генератором для порождения *ключевого потока*. Обычно представляется как некоторая последовательность двоичных чисел.

Ключевой поток (Keystream) – псевдослучайная последовательность двоичных чисел, которая используется для шифрования и дешифрования информации.

Шифртекст (Ciphertext) – последовательность, полученная посредством объединения исходного текста и *ключевого потока*. Обычно в результате сложения по модулю двух соответствующих двоичных последовательностей.

Генератор ключевого потока (Keystream generator) – тотальная дискретная функция, множеством значений входных переменных которой является множество значений секретного ключа, а выходным – ключевого потока.

Brute-force – простейшая криптографическая атака, в основе которой лежит полный перебор всего пространства значений функции для нахождения решения.

Guess-and-determine – подход, основанный на поиске эффективных декомпозиционных представлений пространства поиска. Найденная декомпозиция является эффективной, если для нахождения решения требуется существенно меньше времени, которое затрачивает *brute-force* метод.

1.2. Обоснование актуальности

На данный момент для обеспечения достаточного уровня защиты информации в различных системах и протоколах передачи данных используются криптографические алгоритмы. Для оценки уровня безопасности вводится понятие криптографической стойкости. С помощью разнообразных способов обращения криптографических функций, называемых криптографическими атаками, можно оценить данный параметр.

Любая нетривиальная аналитическая атака на известный криптографический алгоритм является крупным успехом. Однако для построения каждой такой атаки требуется скрупулезный анализ особенностей рассматриваемого алгоритма.

В настоящее время существует множество различных криптографических алгоритмов. Для некоторых из них были построены эффективные атаки, которые позволяют осуществить взлом на современных компьютерах за считанные секунды. Но есть и такие, для которых до сих пор не найдено ни одной атаки, которую можно было бы назвать эффективной. Это обуславливается тем, что подходы, которые успешно показали себя на одних алгоритмах, оказываются совершенно

бесполезными для других. Главным недостатком аналитического метода является необходимость в индивидуальном рассмотрении особенностей каждого алгоритма, что требует больших затрат времени и ручного труда.

Автоматизированные методы, которые будут рассмотрены в этой работе, позволяют строить атаки из класса *guess-and-determine* для многих известных шифров. В основе описываемых методов лежат современные алгоритмы решения задачи булевой выполнимости (*SAT*), поскольку они показывают хорошие результаты в криптоанализе. Для построения атак, основанных на данных методах, на первом этапе требуется свести рассматриваемую задачу криптоанализа к *SAT*-задаче. Следующий этап предполагает решить полученную *SAT*-задачу путем разбиения ее на более простые подзадачи с помощью построенного декомпозиционного множества и последующим решением этих подзадач.

Конечно, автоматизированные методы не призваны полностью заменить аналитические. Они могут быть использованы для первоначальной оценки исследуемого алгоритма. Но, в некоторых ситуациях, этой оценки может быть достаточно для обоснования недостаточной криптографической стойкости.

1.3. Существующие автоматические трансляторы и SAT-решатели

Одним из этапов работы является сведение задачи криптоанализа к задаче о булевой выполнимости. То есть преобразование алгоритма, задающего рассматриваемую криптографическую функцию, в КНФ. Для достижения этой цели существуют специализированные программы – автоматические трансляторы. Рассмотрим подробнее некоторые из них.

Cryptol [1] – функциональный язык для описания криптографических функций. Изначально был разработан для

эффективной реализации криптографических алгоритмов в аппаратных схемах. Поэтому *Cryptol* хорошо подходит для сведения к *SAT* и *SMT* криптографических процедур и протоколов. Несомненным преимуществом является наличие подробной документации. К недостаткам можно приписать неудобную базовую модель вычислений.

URSA [2] – инструмент для пропозиционального кодирования, который применим к широкому классу комбинаторных задач, начиная от задач программирования в ограничения, и заканчивая криптографическими алгоритмами. Для описания используется специальный предметно-ориентированный язык. Недостатком является отсутствие какой-либо документации.

Transalg [3] – система для транслирования дискретных функций в *SAT*. Описание функции производится с помощью процедурного языка программирования *TA*, который имеет *C*-подобный синтаксис и блочную структуру. К его преимуществам можно отнести: вычислительная модель, использующая прямой доступ к памяти, и встроенный модуль минимизации. Недостатком является отсутствие полной документации.

В данной работе было отдано предпочтение автоматическому транслятору *Transalg*, поскольку он позволяет строить более компактные булевы формулы, что продемонстрировано в статье [4].

Когда функция, задающая исследуемый генератор ключевого потока, оттранслирована в *SAT*, наступает следующий этап – решение *SAT*-задачи. Для этих целей используются *SAT*-решатели (или иначе программные средства решения задачи выполнимости). Рассмотрим некоторые из них.

Minisat [5] – минималистичный *SAT*-решатель с открытым исходным кодом. Обладает наградами ряда соревнований *SAT Competition* [6]. Его авторами были выделены следующие ключевые особенности:

1. Простота модификации, которая достигается благодаря хорошей документации, а также продуманности и простоте его строения.
2. Высокая эффективность, которую доказывают награды, полученные на соответствующих соревнованиях.
3. Нацеленность на интеграцию в качестве модуля.

Lingeling [7] – серьезный инструмент для решения задач булевой выполнимости. Является обладателем множества различных наград таких соревнований как: *SAT Competition* [6], *SAT Race*, *Configurable SAT Solver Challenge* (*CSSC'14*) [8]. Имеет более сотни настраиваемых параметров, что позволяет хорошо адаптироваться к решению различных классов задач. Также имеет интерфейс для запуска в многопоточном режиме – *Plingeling*, и параллельный решатель на основе сочетания методов *Cube & Conquer* и *Conflict-Driven Clause Learning* (*CDCL*) [9] – *Treengeling*.

ROKK [10] – модифицированная версия *SAT*-решателя *Minisat*. Состоит из двух компонентов: *SATELite* [11] и, собственно, *Minisat*. Первый компонент осуществляет упрощение решаемой *SAT*-задачи, после чего упрощенная КНФ передается в модуль *Minisat*, который описан выше, для дальнейшего решения.

В данной работе используются все рассмотренные выше *SAT*-решатели. Одной из особенностей программной среды является возможность простого переключения между используемыми *SAT*-решателями, а также добавления новых.

1.4. Обзор исследуемых криптографических алгоритмов

На данный момент существует большое число различных криптографических алгоритмов. Некоторые из них являются очень стойкими к атакам различного рода. Также существуют и такие, который

могут быть взломаны на обычном домашнем компьютере с помощью простого метода перебора.

В данной работе будут рассмотрены генераторы ключевого потока: *A5/1*, *Bivium*, *Trivium 64*. Перечисленные алгоритмы, на данный момент, не являются самыми стойкими, но также и не являются слишком простыми. Их изучение с помощью развиваемых в настоящей работе методов является хорошей отправной точкой для дальнейших исследований.

A5/1 – поточный алгоритм шифрования, который долгое время использовался в европейском стандарте сотовой связи *GSM*. Он состоит из трех регистров сдвига с линейной обратной связью (в дальнейшем *РСЛОС*) с длинами 19, 22, 23 бита соответственно. В течении такта каждый бит перемещается в следующую ячейку, а бит в последний ячейке является выходным. В первую же ячейку записывается бит заранее вычисленный с помощью функции обратной связи. Для шифра *A5/1* обратная связь описывается с помощью следующих многочленнов для каждого регистра соответственно: $X^{19}+X^{18}+X^{17}+X^{14}+1$, $X^{22}+X^{21}+1$, $X^{23}+X^{22}+X^{21}+X^{8}+1$. Управление тактированием осуществляется с помощью специального механизма. Для каждого регистра определены следующие биты синхронизации: $b_1 = 8$, $b_2 = 10$ и $b_3 = 10$ для первого, второго и третьего регистров соответственно. Регистр с номером j сдвигается, истинна следующая булева формула: $b_j \equiv majority(b_1, b_2, b_3)$.

Теперь рассмотрим шифр *Trivium*, поскольку рассматриваемые далее генераторы ключевого потока *Bivium* и *Trivium 64* являются его упрощениями.

Trivium – симметричный алгоритм синхронного поточного шифрования. Был представлен в 2008 году в рамках проекта *eSTREAM* [12]. Он состоит из 3 сдвиговых регистров с длинами 93, 85, 110 бит соответственно. На каждом такте происходит синхронный сдвиг всех

регистров, причем бит обратной связи замещает первый бит в соседнем регистре, а не в своем. То есть бит обратной связи первого регистра t_1 используется вторым регистром, t_2 – третьим регистром и t_3 – первым регистром. Также данный алгоритм имеет фазу инициализации. Состояние регистров инициализируется с помощью двух векторов длиной 80 бит: секретного ключа и открытого вектора инициализации, которые частично заполняют первый и второй регистры соответственно. После этого совершается 4 полных прохода алгоритма, то есть 1152 такта. Гарантируется что состояние всех регистров будет полностью зависеть от исходных векторов. Данный алгоритм шифрования довольно сложен, поэтому для исследования были выбраны шифры *Bivium* и *Trivium 64*, которые принадлежат семейству *Trivium*.

Trivium 64 – симметричный алгоритм синхронного поточного шифрования полученный из алгоритма *Trivium* путем уменьшения длин регистров до 22, 19, 23 битов соответственно. Способ сокращения длин регистров, который использовался для данной модификации, описан в статье [13] и его применение позволяет сохранять алгебраические свойства исходного шифра. Уменьшение длин регистров позволило существенно снизить его сложность. Система тактирования осталась без изменений.

Bivium – шифр, который отличается от алгоритма *Trivium* тем, что содержит только два сдвиговых регистра с длинами 93 и 84 бита. Хоть суммарная длина его ключа и составляет 177 бит, сам алгоритм намного слабее *Trivium*. Система тактирования и фаза инициализации не изменились.

1.5. Обзор существующих автоматизированных методов

Теперь можно перейти к рассмотрению существующих автоматизированных методов, которые позволяют строить

декомпозиционные представления сложных *SAT*-задач с лучшей оценкой трудоемкости решения. В данном разделе будут описаны два различных метода построения атак из класса *guess-and-determine*: *SAT Partitioning* (*SAT*-разбиение) и *Inverse Backdoors Set* (*IBS*). Одним из главных отличий этих методов является то, что они находят декомпозиционные множества разного рода. *SAT Partitioning* позволяет находить множества специализирующиеся на доказательстве невыполнимости *SAT*-задачи, то есть доказательстве того, что не существует такого множества значений переменных, при которых формула принимает значение *истина*. *IBS* же, напротив – на доказательстве выполнимости *SAT*-задачи.

И тот, и другой подходы показывают хорошие результаты на реальных шифрах.

1.5.1. Атаки на основе *SAT*-разбиений

В статье [14] описан параллельный алгоритм для поиска хороших декомпозиционных представлений (*good partitioning*) исследуемой SAT-задачи, то есть с лучшей оценкой трудоемкости решения. Также описан и метод нахождения оценки для данной декомпозиции, который позволяет сравнивать множества между собой. Для поиска используются метаэвристические алгоритмы: *поиск с запретами* и *имитация отжига*. Разработанные авторами алгоритмы были применены для построения *guess-and-determine* атак на такие известные алгоритмы, как *A5/1* и *Bivium*.

Для начала опишем процесс представления декомпозиционного множества в виде булева вектора. Пусть X – множество всех переменных рассматриваемой *SAT*-задачи, $X \subset \overline{X}$ – декомпозиционное множество, тогда если переменная $x_i \in \overline{X}$, то $\chi_i = 1$, иначе $\chi_i = 0$. Теперь каждая декомпозиция может быть представлена как булев вектор $\chi = (\chi_1, \ldots, \chi_n)$,

$n = |X|$. Далее описывается метод оценивания трудоемкости решения полученных векторов, что позволит сравнивать их между собой.

Для вычисления этого значения положим, что C – исходная КНФ, а G_j – подстановка переменных из декомпозиционного множества, где $j \in \{1, \ldots, 2^s\}$, а s – мощность декомпозиции. Тогда значение может быть найдено, как сумма времен работы *SAT*-решателя на всех возможных подстановках из декомпозиционного множества, то есть на всех *КНФ*: C G_j. Поскольку зависимость имеет экспоненциальный характер а решение задачи выполнимости для каждой отдельной КНФ вида C G_j может потребовать существенного времени, временные затраты на нахождение только одного значения могут быть колоссальными. Поэтому будет производиться не вычисление значения, а его оценка с использованием метода Монте Карло [15]. В соответствии с данным методом, для конкретного декомпозиционного множества $\overline{X}$ строится случайная выборка объема N значений переменных из $\overline{X}$. Через ξ_j обозначим время работы *SAT*-решателя на получаемой КНФ вида C G_j. Значение функции, оценивающей суммарную трудоемкость обработки соответствующего *SAT*-разбиения выглядит следующим образом:

$$F = 2^s \frac{1}{N} \sum_{1}^{N} \xi_j .$$

Тем самым задачи построения *SAT*-разбиения с наименьшим временем обработки и, как следствие, построение лучшей *guess-and-determine* атаки, сводится к минимизации функции F. Поскольку данная функция не может быть задана аналитически, ее минимизация будет осуществлять с помощью метаэвристических алгоритмов. Авторами [14] были использованы: *поиск с запретами* и *имитация отжига*, причем первый показал лучшие результаты.

1.5.2. Атаки на основе Inverse Backdoor Set

Особенностью данного подхода является нахождение так называемых *backdoor* множеств, которые позволяют разложить трудную задачу на более простые. То есть нахождения эффективных декомпозициоых представлений. Техника построения таких множеств, описанная в статье [16], была специально разработана для решения задач обращения криптографических функций. Алгоритм был применен к следующим блочным и поточным шифрам: *Trivium*, *AES-128*, *Magma* (*ГОСТ 28147-89*).

Поскольку *backdoor* множества также состоят из переменных рассматриваемой *SAT*-задачи, метод их представления в виде булевого вектора ничем не отличается от описанного выше. Перейдем к рассмотрению функции для оценки трудоемкости решения.

Для начала необходимо построить пару, состоящую из секретного ключа и ключевого потока, для рассматриваемого криптографического генератора. Пусть C – исходная КНФ, тогда подставив в нее значения переменных ключевого потока γ получим задачу криптоанализа $C(\gamma)$. Также подставим значения только тех переменных секретного ключа χ, которые находятся в оцениваемом *backdoor* множестве. Подсчитаем вероятность того, что случайно выбранная задача из всего множества для данной декомпозиции будет решена с ограничением времени t: $P_B = \frac{\#\{\chi \in \{0,1\}^n : T(C(\gamma_\chi, \chi)) \le t\}}{2^n}$.

Рассмотрим последовательность ключевый потоков $\gamma_1, \ldots, \gamma_r$, тогда вероятность обратить хотя бы один из них: $P = 1 - (1 - P_B)^r$, а значение оценки трудоемкости будет выглядеть следующим образом: $F = 2^s \cdot t \cdot r$, где s – мощность *backdoor* множества. Несложно показать, что $P > 0.95$ при $r \ge \frac{3}{P_B}$. Осталось только оценить значение P_B, поскольку его

вычисление требует больших временных ресурсов. С помощью метода Монте-Карло [15] приходим к формуле:

$P_B = \frac{1}{N} \sum_{i=1}^{N} \xi_i$, где $\xi_i = 1$ если задача была решена за отведенное время, и $\xi_i = 0$ иначе.

Для минимизации полученной оценочной функции в статье [16] был использован метаэвристический алгоритм *поиск с запретами*.

1.6. Постановка исследовательского задания

В вычислительных экспериментах, представленных в статьях [14] и [16], использовался мощный вычислительный кластер (10 узлов с двумя процессорами *Intel Xenon E5-2695 v4* [17]), что сгладило некоторые недостатки использованных метаэвристических схем. В данной работе в качестве альтернативы будут использованы эволюционные алгоритмы: различные эволюционные стратегии и генетический алгоритм. Они будут являться основой разрабатываемых алгоритмов автоматизированного построения декомпозиционных множеств для задач криптоанализа.

Также ввиду ограничений на использование вычислительных ресурсов будут предложены различные эвристики, которые позволят уменьшить число производимых вычислений и существенно ускорить алгоритм без особого влияния на конечный результат.

В данной работе будет реализована программная среда для автоматизированного построения *guess-and-determine* атак, основанных на алгоритмах решения *SAT*, на криптографические функции. Будет предложена стратегия адаптивного изменения объема выборки в зависимости от значения оценочной функции. Также будут построены новые декомпозиционные множества для ряда современных шифров: *Trivium 64*, *A5/1*, *Bivium*.

Выводы по главе 1

В данной главе были приведены основные термины и определения исследуемой предметной области. Также были рассмотрены используемые программные средства: автоматические трансляторы и *SAT*-решатели. Был произведен обзор исследуемых шифров и существующих автоматизированных методов построения *guess-and-determine* атак. Была приведена постановка исследовательского задания.

Глава 2. Разработка и реализация автоматизированных методов анализа криптографических функций

В предыдущей главе были рассмотрены автоматизированные методы построения декомпозиционных представлений трудных задач о булевой выполнимости. В рамках этих методов был описан способ представления декомпозиционного множества в виде булевого вектора, а также были введены оценочные функции для построения *guess-and-determine* атак на основе *SAT Partitioning* и *Inverse Backdoor Set*.

В данной главе будут рассмотрены этапы разработки и реализации алгоритмов автоматизированного построения декомпозиционных представлений задач криптоанализа.

2.1. Разработка общей схемы

В основе разрабатываемого метода лежит стандартная схема *эволюционного алгоритма* (*evolutionary algorithm*), которая состоит из нескольких фаз и представлена на рисунке 1. В качестве функции приспособленности выступают описанные выше оценочные функции на основе стратегий *SAT Partitioning* и *Inverse Backdoor Set*.

В качестве *особи* (*individual*) выступает декомпозиционное множество представленное в виде булевого вектора, которое в дальнейшем подвергается изменениям с помощью *мутации* (*mutation*) и *кроссинговера* (*crossover*), которые свойственны эволюционным алгоритмам. Выбор функции мутации и кроссинговера для исследуемой задачи будет рассмотрен в следующей разделе.

Особенностью эволюционного алгоритма является наличие так называемых *эволюционных стратегий* (*evolution strategy*), а также его

изменения до *генетического алгоритма* (*genetic algorithm*). Основными являются следующие стратегии:

(*m*, *l*) – размер популяции равен *l*, после вычисления значения функции приспособленности выбираются *m* особей и с помощью функции мутации получают *l* новых особей, которые формируют следующую популяцию.

(*m* + *l*) – отличается от стратегии (*m*, *l*) тем, что новая популяция формируется из *m* выбранных особей и только *m* - *l* получают посредством использования функции мутации.

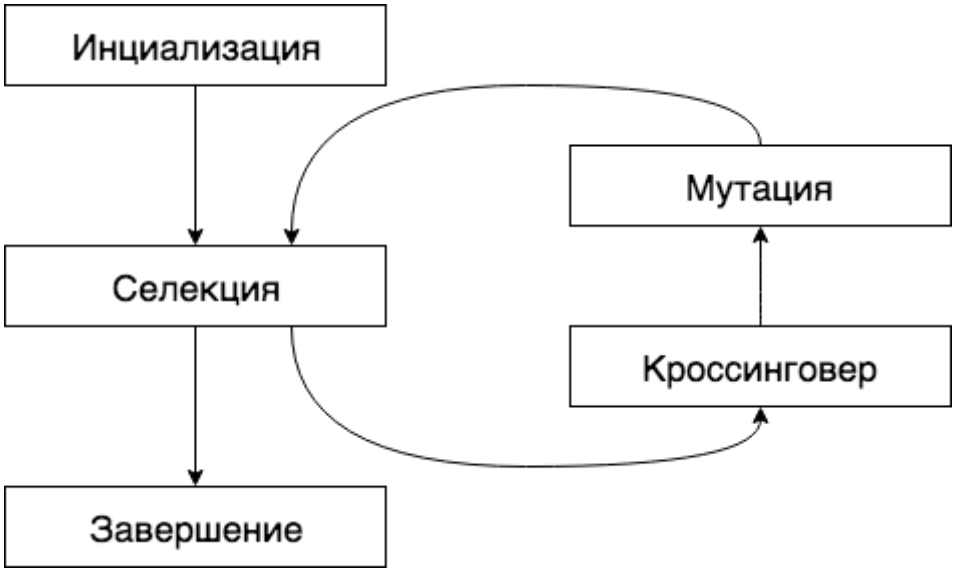

Рисунок 1. Схема эволюционного алгоритма.

Генетический алгоритм отличается от эволюционных схем тем, что для изменения особей, помимо функции мутации, используется кроссинговер.

Теперь рассмотрим подробнее фазы эволюционного алгоритма.

В фазе *инициализации* (*initialization*) создается первая популяция из *l* случайный особей. Данная популяция будет являться первым поколением.

В фазе *селекции* (*selection*) для каждой особи из рассматриваемой популяции будет подсчитано значение функции приспособленности. После подсчета выбираются *m* лучших особей. Выбор происходит путем нахождения особей с максимальными значениями.

В фазе *мутации* (*mutation*) и *кроссинговера* (*crossover*) подготавливается новая популяция, путем создания *потомков* (*child*) особей, полученных на предыдущей фазе. Соответствующие функции будут рассмотрены в следующем разделе.

Завершение работы алгоритма происходит путем выполнения некоторого условия. Условием может являться достижение некоторого значения оценочной функции, исчерпание лимита на вызов этой функции или нахождения локального минимума.

Для вычислительных экспериментов предполагается рассмотреть стратегии (1 + 1), (1 + 2), (1 + 5) и генетический алгоритм.

2.2. Подбор параметров и функций

Одним из важных этапов разработки алгоритма является подбор параметров и функций для рассматриваемых эволюционных стратегий и генетического алгоритма. Их подбор осуществлялся с использованием стратегии (1 + 1) и проверялся на оставшихся.

В первую очередь была подобрана функция мутации, которая свойственна эволюционным алгоритмам. Поскольку особь представлена в виде битового вектора, который задает соответствующее декомпозиционное множество, то были рассмотрены соответствующие функции.

Изначально была опробована функция одиночного инвертирования бита, то есть среди всех битов случайно выбирается один и его значение заменяется противоположным. Хоть эта функция и является простой и

очевидной, с помощью неё были построены неплохие декомпозиционных множества, с которых можно было начать исследование. Следующей была функция равномерной мутации, также известная как *bit-flip mutation*. Ее принцип тоже довольно прост: каждый бит вектора инвертируется с некоторой вероятностью. Часто эта вероятность равна $\frac{1}{l}$, где l – длина соответствующего вектора. Математическое ожидание числа измененных битов в данном случае будет равно одному. Главное отличие в том, что первая функция изменяет один и только один бит за раз, а вторая, в свою очередь, с ненулевой вероятностью может изменить два и даже три бита за раз. Благодаря этому, появляется вероятность перейти из одного локального пространства в другое. Поскольку эта вероятность довольно мала, то такой переход будет происходить редко. Он будет полезен в случае долгого простаивания в одной из точек пространства.

Для рассматриваемой задачи специфичен довольно быстрый спуск на первых итерациях до определенного значения. После этого алгоритм пытается улучшить построенное декомпозиционное множество и зачастую долго простаивает в какой либо точке. Описанная выше функция мутации *bit-flip* хорошо подходит для решаемой задачи. А также доказала свою эффективность в ходе проведения экспериментов.

Важным параметром является размер выборки, на котором рассчитывается значение оценочной функции. От его величины зависит точность значения, полученного при вычислении оценочной функции. Было экспериментально установлено, что довольно хорошие оценки достигаются при значениях порядка нескольких сотен. Для проведения дальнейших экспериментов был выбран объем выборки, равный 1000. Но дальнейшие исследования показали, что при подсчете значений на некоторых декомпозиционных множествах будет достаточно и небольшой

выборки. Рассмотрению данного вопроса посвящен четвертый раздел настоящей главы.

Следующим важным значением является число стагнаций. Данный параметр свойственен эволюционным алгоритма и больше всего зависит от выбранной эволюционной стратегии. Этот параметр позволяет определить оптимальное число стагнационных итераций, после которых необходимо сделать рестарт. Также он зависит от характера рассматриваемой задачи, поэтому было принято решение сделать его внешним настраиваемым параметром. Для экспериментов были подобраны следующие оптимальные значения: для стратегии (1+1) – 100, стратегии (1 + 2) – 50 и для стратегии (1 + 5) и генетического алгоритма – 20. На самом деле необходимо было определить значение только для стратегии (1 + 1), поскольку, например, при использовании стратегии (1 + 2) область поиска увеличивается вдвое по сравнению с (1 + 1). Исходя из этого можно предположить, что на нахождения нового значения потребуется вдвое меньше итераций. Аналогичные рассуждения будут также справедливы и для стратегии (1 + 5).

2.3. Особенности алгоритма

В этом разделе будут описаны некоторые особенности разрабатываемой программной среды, использование которых существенно повлияло на процесс проведения экспериментов и полученные результаты.

Одной из таких особенностей является возможность выбора *SAT*-решателя из списка уже имплементированных в среду, с помощью которого будет считаться значение оценочной функции, а также простой механизм добавления новых программных средств решения задачи выполнимости. На данный момент в этом списке находятся *SAT*-решатели,

описанные в первой главе, а именно: *Minisat*, *ROKK*, *Lingeling* и его параллельные версии *Treengeling* и *Plingeling*. Благодаря этой особенности удалось заметить, что значения оценочной функции, вычисляемые в процессе поиска эффективного декомпозиционного множества, зависят от используемого *SAT*-решателя, причем нельзя выделить однозначного лидера. Однако в процессе экспериментов удалось для каждого из рассматриваемых генераторов подобрать *SAT*-решатель, показывающий лучшую эффективность. Скорее всего, это происходит из-за того, что каждое из рассматриваемых программных средств в процессе поиска решения использует свои уникальные эвристики. А каждый генератор ключевого потока имеет свои особенности, вытекающие из его строения, алгебраических свойств, а также способа транслирования в *SAT*.

В таблице 1 приведен пример вычисления оценочной функции для одного и того же декомпозиционного множества с использованием разных *SAT*-решателей. Исследуемый генератор ключевого потока – *A5/1*, размер выборки – 10000.

Таблица 1. Пример вычисления оценочной функции разными *SAT*-решателями.

Решатель	Число решившихся	Процент решившихся	Значение
ROKK	11	0,11 %	$1{,}49 \cdot 10^{13}$
Lingeling	57	0,57 %	$2{,}84 \cdot 10^{12}$

Также в процессе поиска используется хэширование значений вычисленных оценочных функций. Данная модификация позволяет увеличить число различных пройденных точек пространства поиска, поскольку вычисление оценочной функции в каждой точке требует существенного времени. А также полностью оправдала себя при

проведении экспериментов. Более 60 % запросов на вычисление значения оценочной функции было покрыто хэшем. Приведем статистику хэшируемости для разных эволюционных стратегий и генетического алгоритма в таблице 2.

Таблица 2. Статистика хэшируемости для различных стратегий

	Число запросов		
Стратегия	Всего	Вычислено	Взято из хеша
(1 + 1)	~560	~160	71 %
(1 + 2)	~800	~300	63 %
(1 + 5)	~1200	~470	61 %
ГА (10)	~3020	~680	77 %

2.4. Адаптивное изменение объема выборки

В ходе проведения экспериментов были проанализированы полученные данные и выявлена следующая закономерность. Алгоритм начинает свою работу с декомпозиционного множества, которое содержит все множество переменных и в процессе работы исключает из него менее релевантные. Пока множество содержит большое число переменных, его обновление происходит почти на каждой итерации. Вследствие этого на графиках, представленных на рисунке 2, происходит резкий спуск на первых итерациях. На последующих итерациях алгоритм начинает дольше простаивать в некоторых точках и процесс улучшения значения функции замедляется.

Также, чем больше декомпозиционное множество, тем проще получаются подзадачи полученные подстановкой переменных из этого

множества. Из за этого процент успешно решенных задач с заданным ограничением по времени при построении *guess-and-determine* атак на основе *IBS* довольно высок при большом числе переменных.

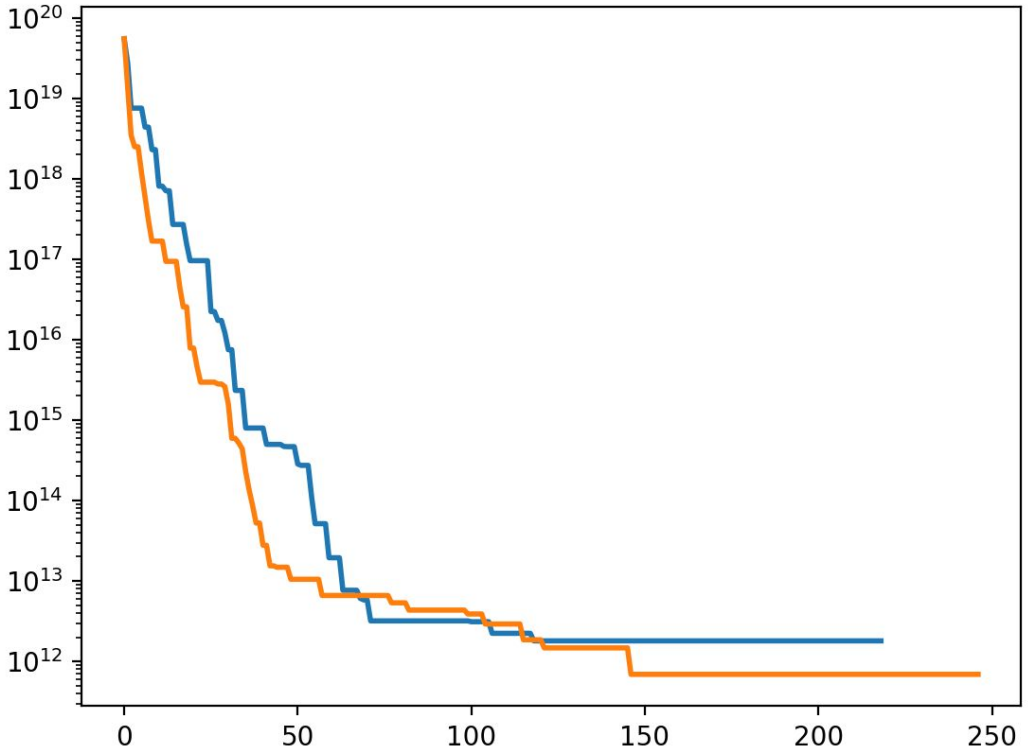

Рисунок 2. Показательные резкие спуски на первых итерациях

Вкупе эти две закономерности позволяют предложить улучшение, которое позволит существенно сократить время затрачиваемое алгоритмом на спуск до определенного значения. В данном случае в качестве границы можно использовать значение оценочной функции, поскольку оценка эффективных декомпозиционных множеств одной мощности разнится в разумных пределах.

Суть предлагаемого улучшения заключается в том, чтобы изменять объем выборки в процессе работы алгоритма в зависимости от текущего значения оценочной функции. Причем вычисления начинаются с выборки размером 10 элементов и изменяются до 50, 100, 300, 500, 800 и 1000 в

порядке возрастания. Переход к следующему размеру выборки происходит при достижении значением оценочной функции некоторой границы.

Для определения этих границ были проведены эксперименты, которые будут подробнее описаны в следующей главе. Результаты же представлены в таблице 3. Полученные граничные значения являются эмпирическими величинами. Зависимости представленные в талицы является только лишь следствием проведенных экспериментов. Для каждого генератора ключевого потока эти граничные значения будут разными, поскольку напрямую зависят от его сложности, а также мощности множества по которому строится декомпозиция, иначе говоря множество битов секретного ключа.

Таблица 3. Эмпирическая зависимость для шифра *A5/1* объема выборки, достаточного для получения относительно точного значения оценочной функции.

Объем выборки	Значение оценочной функции
10	до $5{,}5 \cdot 10^{17}$
50	до $4{,}9 \cdot 10^{16}$
100	до $8{,}1 \cdot 10^{14}$
300	до $1{,}1 \cdot 10^{14}$
500	до $1{,}7 \cdot 10^{13}$
800	до $6{,}7 \cdot 10^{12}$
1000	после $6{,}7 \cdot 10^{12}$

Данная модификация позволит алгоритму доходить до граничного значения с явным выигрышем по времени. Вопрос о выигрыше времени,

затрачиваемого алгоритмом на поиск минимального значения, остается открытым, поскольку заранее нельзя точно предсказать, сколько потребуется совершить итераций для его достижения.

В таблице 4 будет продемонстрировано сравнение затрат временных ресурсов для достижения граничного значения для алгоритма с данной модификации и алгоритма без нее. Сравнение будет проводится для всех заявленных к рассмотрению стратегий, то есть (1 + 1), (1 + 2), (1 + 5) и генетического алгоритма.

Подобная идея не применялась ранее для построения *guess-and--determine* атак на криптографические алгоритмы. Конечно, необходимо произвести больше исследований и выработать более гладкую стратегию для изменения объема выборки, но даже в таком виде по результатам представленным в таблице 4 можно сделать вывод, что выигрыш по времени является существенным, и это позволяет сказать о ее успешном применении в данной области. Для большей наглядности эти же результаты представлены на рисунке 3 в виде диаграммы.

Таблица 4. Результаты адаптивного метода для различных стратегий

	Наличие адаптивного изменения объема выборки	
Стратегия	С	Без
(1 + 1)	~1ч	~2ч
(1 + 2)	~1ч 6м	~2ч 30м
(1 + 5)	~1ч 17м	~2ч 50м
ГА (10)	~2ч 50м	~4ч 40м

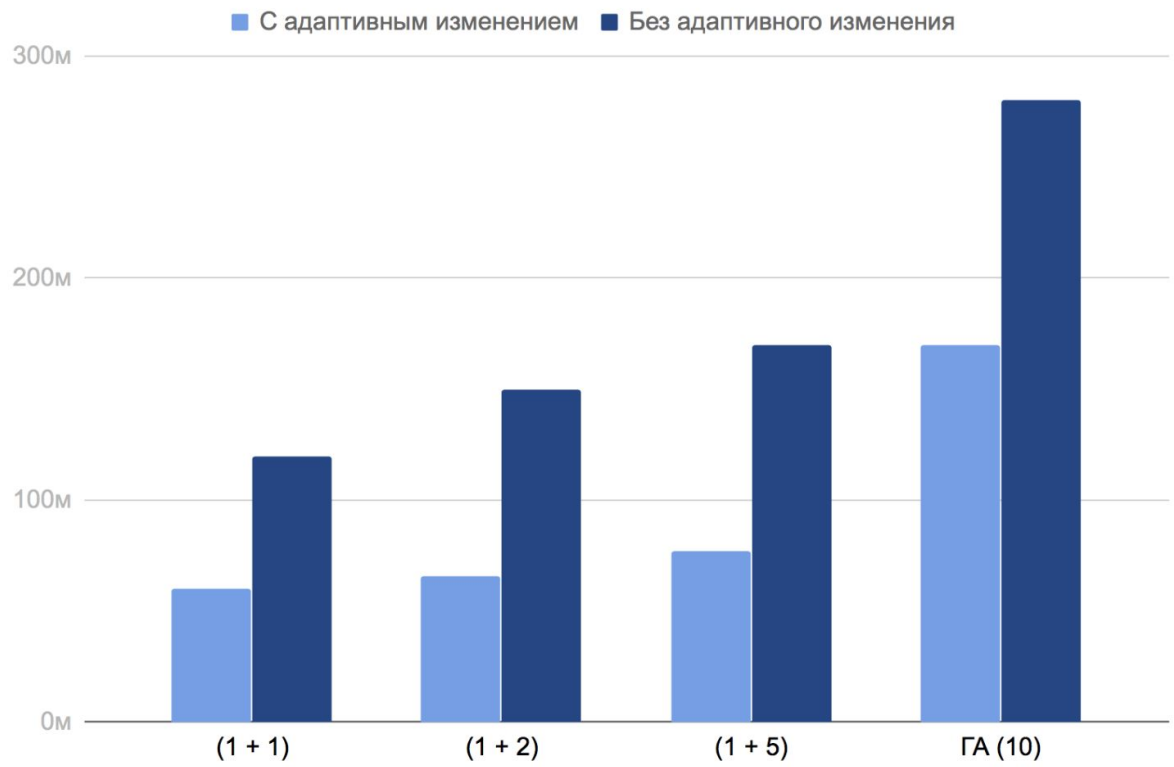

Рисунок 3. Сравнение временных затрат на достижение граничного значения для разных эволюционных стратегий и генетического алгоритма

2.5. Реализация алгоритма

В данном разделе будет приведен псевдокод основных элементов разрабатываемой программной среды.

В листинге 1 представлен класс реализующий эволюционной алгоритм. Алгоритм начинает свою работу с инициализации стартовой популяции. Затем пока не будет достигнуто условие остановки происходит смена поколений для поиска лучшей особи. Для каждой особи в поколении либо вычисляется значение оценочной функции. Если значение уже было вычислено ранее, то оно восстанавливается из хеша. После того как были подсчитаны все значения происходит выбор лучшей особи из поколения, и ее дальнейшее сравнение с лучшей особью текущего запуска. Когда лучшая особь была определена происходит смена поколения. Если условие

остановки будет выполнено, результатом работы алгоритма будет являтся лучшая особью на момент завершения.

Листинг 1. Класс эволюционного алгоритма.

```
class EvolutionaryAlgorithm():
    start():
        population = strategy.get_start_population()
        while not stop_condition:
            for p in population:
                if p in hash:
                    value = hash[p]
                else:
                    N = adaptive_selection.get()
                    value = compute_value(p, N)
                p.value = value
                p_best = get_best(population)
                if best > p_best:
                    best = p_best
            population = strategy.get_new_population(
                                    population)
        return best
```

В листинге 2 представлен метод для подсчета значения оценочной функции. Вначале генерируется выборка размера *N* из инициализирующих подзадач. Затем создаются `worker's` в количестве определенном ранее и происходит их запуск. Главный поток ждет пока все `worker's` не будут завершены, после чего подсчитывает значение оценочной функции.

Листинг 2. Метод для подсчета значения оценочной функции для построения *guess-and-determine* атак на основе *IBS* метода.

```
def compute_ibs_value(X, N)
    init_cases = generate_init(N)
    solved_cases = []
    workers = create_workers(tread_count)
    for worker in workers:
        worker.start()
    while exist alive worker:
        sleep()
    return get_value(solved_cases)
```

В листинге 3 представлен класс `Worker`, реализующий интерфейс `Thread`, используется для параллельного запуска *SAT*-решателей на множестве подзадач. После запуска каждый `worker` берет инициализирующую задачу из заранее подготовленного списка. Если задач не осталось – завершается. Для синхронизации используются методы `lock()` и `release()`. Инициализирующую задача решается методом *Unit Propogation* для получения значений *секретного ключа* и *ключевого потока*. Используя найденные значения и декомпозиционное множество формируется подзадача. Полученная подзадача решается с ограничением по времени. Решенные подзадачи помещаются в отдельный список для дальнейшего подсчета значения оценочной функции.

Листинг 3. Класс, использующийся для параллельного решения подзадач.

```
class Worker(Thread):
    run():
        while True:
```

```
init_cases.lock()
if len(init_cases) > 0:
    init_case = init_cases.pop()
    init_cases.release()
    case = prepare_case(init_case, X)
    solved_case = solver.solve(case, tl)
    solved_cases.lock()
    solved_cases.append(solved_case)
    solved_cases.release()
else:
    init_cases.release()
    break
```

Выводы по главе 2

В данной главе была описана общая схема разрабатываемого эволюционного алгоритма и его особенности, которые свойственны для данной задачи. Был описан процесс подбора различных параметров и функций, свойственных эволюционным алгоритмам. Была представлена новая эвристика, которая позволяет сократить время для построения атак из класса *guess-and-determine*, а также наглядно продемонстрировано ее успешное применение для исследуемой задачи. Также был приведен псевдокод основных элементов разрабатываемой программной среды.

Глава 3. Вычислительные эксперименты

В предыдущей главе был описан процесс разработки и реализация эволюционного алгоритма для автоматизированного построения guess-and-determine атак на криптографические функции. В этой главе будут описаны эксперименты, которые были проведены с использованием реализованной программной среды.

Во всех проведенных экспериментах использовалась оценочная функция для построения *guess-and-determine* атак на основе *IBS*. Благодаря особенностям данной функции можно оценить верхнюю границу времени, которое потребуется на ее вычисление.

3.1. Эксперименты с шифром *A5/1*

В данном разделе будет описан порядок действий для нахождения граничных значений оценочной функции для адаптивного изменения объема выборки на примере шифра A5/1. Также будут приведены промежуточные данные и графики для большей наглядности. Этот метод также применим к другим генераторам ключевого потока, что будет продемонстрированно в следующих разделах. Однако промежуточные данные и графики уже не будут приведены.

Для начала определим граничные значения оценочной функции, до которых эти значения незначительно отклоняются от исходных при выбранном объеме выборки. После этого используя полученные границы протестируем метод адаптивного изменения объема выборки и сравним результаты, полученные с применением данной эвристики и без ее применения.

Для определения описанных выше граничных значений было произведено несколько запусков алгоритма на стратегии (1 + 1) с

величиной выборки равной 1000. Для каждого из них были построены графики, на которых красной линией отображена зависимость лучшего значения оценочной функция от текущей итерации, бирюзовым отмечены возможные точки которые были бы получены при подсчете значения оценочной функции на меньшей выборке N_s, и пунктирные линии являются верхним и нижним квартилями для множества возможных точек. Для каждой итерации было вычислено 1000 возможных точек на меньшей выборке N_s. Вычисление происходило следующим образом: для каждой итерации бралось лучшее значение и рассматривалось множество подзадач, на котором оно было подсчитано, из этого множества выбиралось N_s точек 10000 раз и вычислялось 10000 значений оценочной функции на этих подмножествах. Каждое полученное значение отображалось на графике бирюзовой точкой для соответствующей итерации. Серия графиков для одного из запусков алгоритма представлена на рисунках 5 и 6. Благодаря полученным данным удалось построить экспериментальный график, который представлен на рисунке 7, отображающий зависимость объема выборки от граничного значения оценочной функции. По этому графику были получены эмпирические граничные значения оценочной функции, значения которых приведены в таблице 3 предыдущей главы.

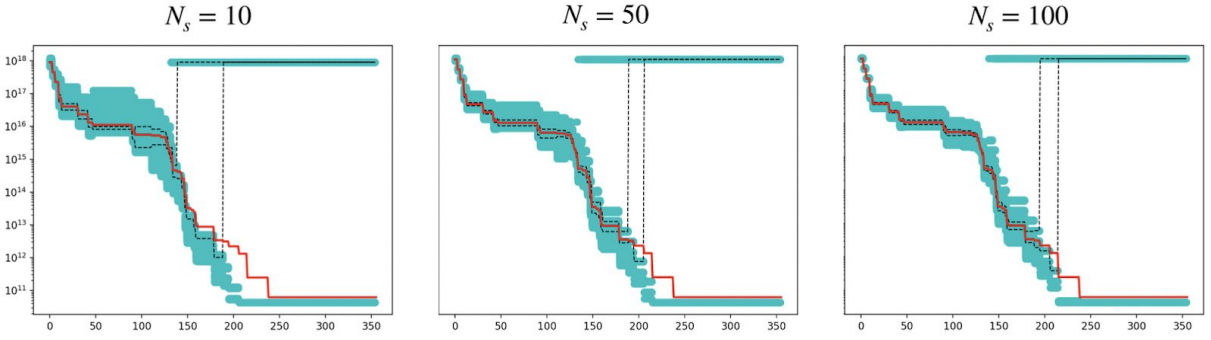

Рисунок 5. Графики, отображающие область возможных значений, для подмножеств мощностью 10, 50 и 100.

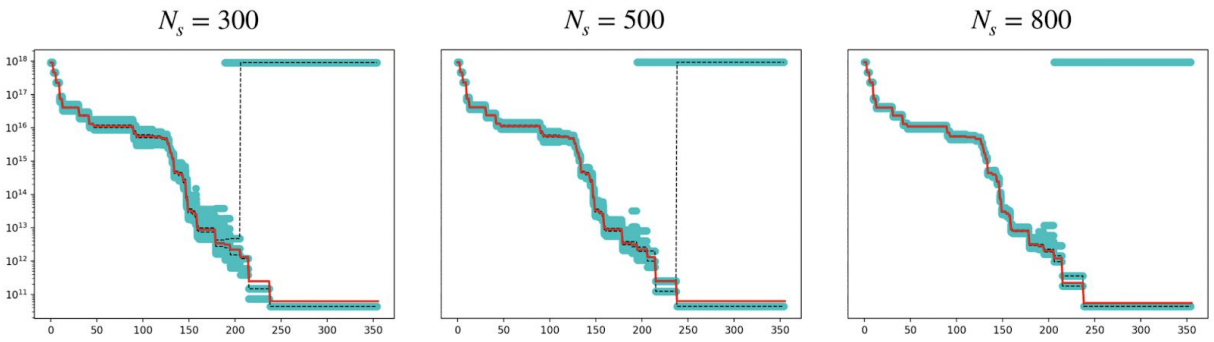

Рисунок 6. Графики, отображающие область возможных значений, для подмножеств мощностью 300, 500 и 800.

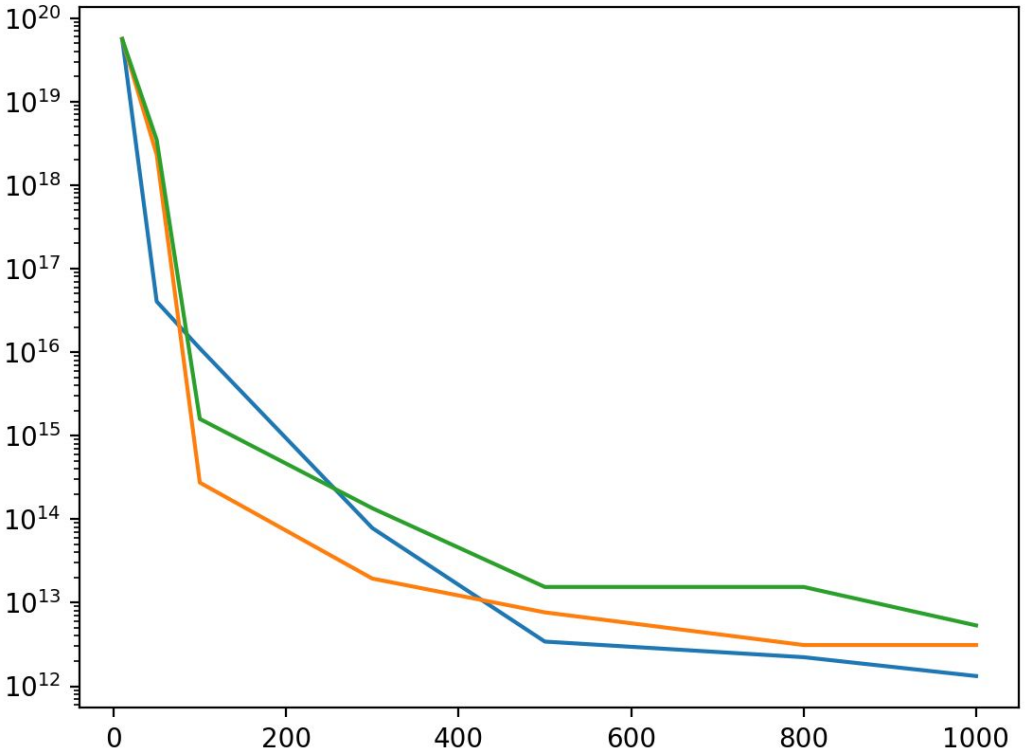

Рисунок 7. Эмпирическая зависимость объема выборки от граничного значения оценочной функции для шифра *A5/1*

После определения описанных выше граничных значений были произведены запуски алгоритма с использованием адаптивного изменения объема выборки. Во всех экспериментах наблюдался предсказанный быстрый спуск на первых итерациях. Один из графиков отображен на рисунке 8. Данные, полученные в этом эксперименте, наглядно

продемонстрировали существенное уменьшение времени, которое тратится на спуск на первых итерациях. Эти данные были представлены в таблице 4 предыдущей главы.

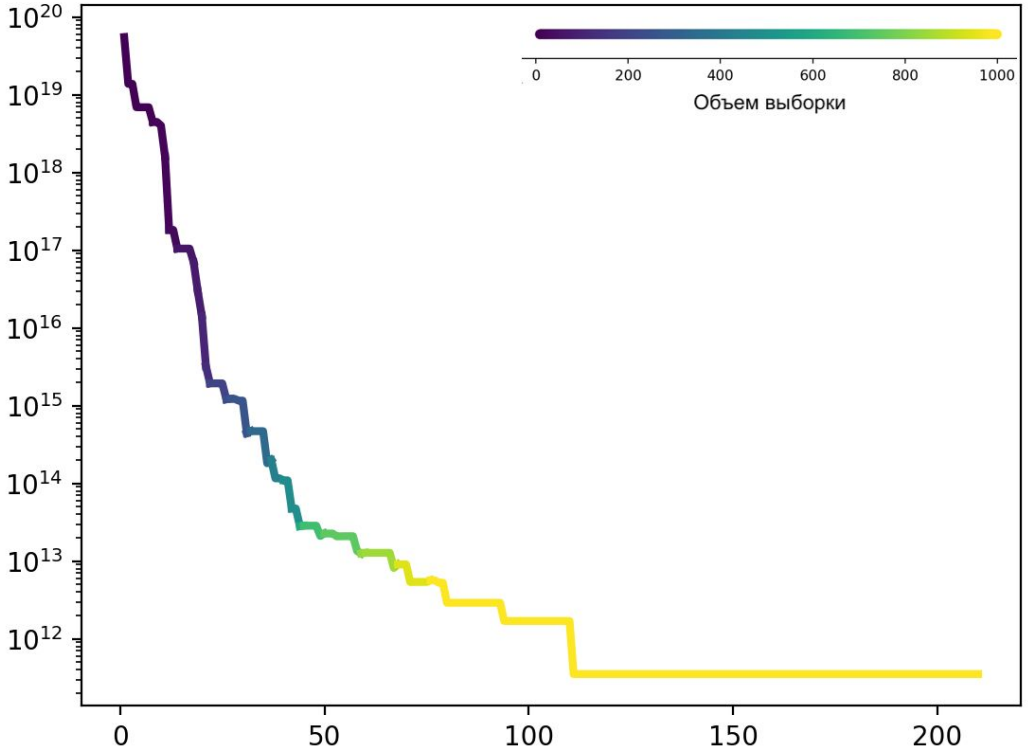

Рисунок 8. Пример запуска алгоритма с адаптивным изменением объема выборки

3.2. Эксперименты с шифром *Trivium 64*

Для данного генератора ключевого потока был произведен эксперимент, описанный в предыдущем разделе, в результате которого был построен график зависимости объема выборки от граничного значения оценочной функции. Этот график приведен на рисунке 9, а граничные значения в таблице 5. По полученным данным можно сделать вывод, что для построения *guess-and-determine* атак на шифр *Trivium 64* достаточно выборки в 300 элементов. Это объясняется тем, что полученные декомпозиционные множества имеют достаточно большой процент решенных подзадач (~60 %), в отличие от множеств, построенных для

шифра *A5/1* (~1 %). При выборке в 300 элементов достигается достаточная точность, чтобы проводить корректные сравнения между разными декомпозиционными множествами для данного шифра.

Таблица 5. Эмпирическая зависимость для шифра *Trivium 64* объема выборки, достаточного для получения относительно точного значения оценочной функции.

Объем выборки	Значение оценочной функции
10	до $4{,}6 \cdot 10^8$
50	до $3{,}1 \cdot 10^8$
100	до $5{,}6 \cdot 10^7$
300	после $5{,}6 \cdot 10^7$

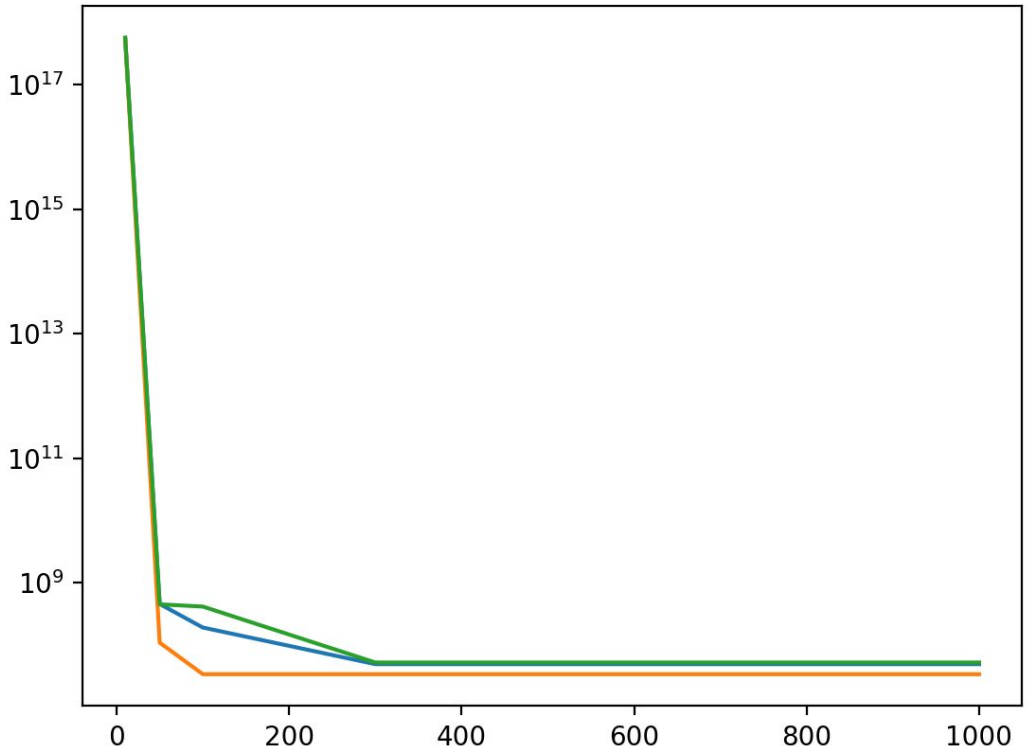

Рисунок 9. Эмпирическая зависимость объема выборки от граничного значения оценочной функции для шифра *Trivium 64*

3.3. Эксперименты с шифром *Bivium*

Для данного шифра также был произведен эксперимент, описанный в первом разделе данной главы, в результате которого был построен график зависимости объема выборки от граничного значения оценочной функции. Этот график приведен на рисунке 10, а граничные значения в таблице 6.

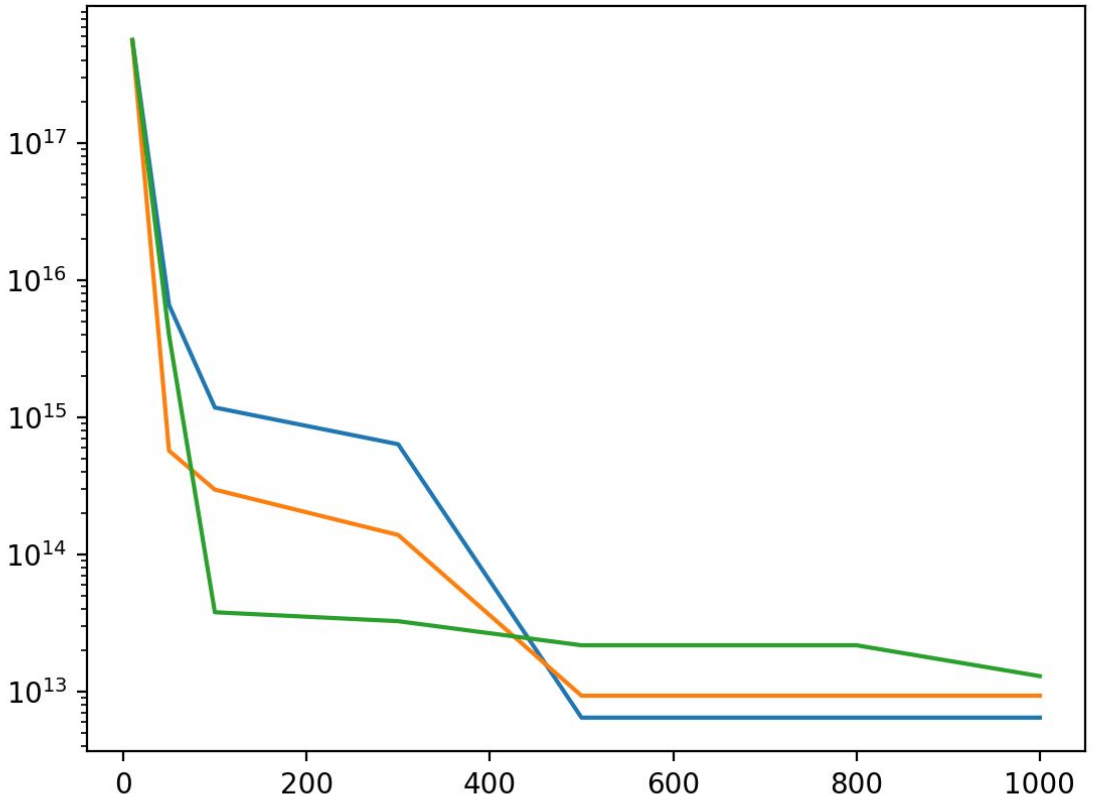

Рисунок 9. Эмпирическая зависимость объема выборки от граничного значения оценочной функции для шифра *Bivium*

Таблица 6. Эмпирическая зависимость для шифра *Bivium* объема выборки, достаточного для получения относительно точного значения оценочной функции.

Объем выборки	Значение оценочной функции
10	до $6{,}6 \cdot 10^{15}$
50	до $1{,}8 \cdot 10^{15}$
100	до $6{,}3 \cdot 10^{14}$
300	до $3.3 \cdot 10^{13}$
500	до $2.2 \cdot 10^{13}$
1000	после $2.2 \cdot 10^{13}$

3.4. Обсуждение результатов

В данном разделе будет произведено сравнение результатов полученных разработанным в данной работе алгоритмом и алгоритмом, предложенным авторами статей [14] и [16]. Основное отличие предложенных методов состоит в том, что для минимизации значения оценочной функции используются разные метаэвристические схемы.

Для построения *guess-and-determine* атак авторами статей [14] и [16] был использован мощный вычислительный кластер (10 узлов с двумя процессорами *Intel Xenon E5-2695 v4* [17]). Вычислительные эксперименты описанные в данной главе проводились на существенно менее мощном сервере (½ процессора *AMD Opteron 6378* частотой *2.4 GHz*). Несмотря на значительный перевес в вычислительных мощностях (примерно в 10 раз) удалось построить более эффективную guess-and-determine атаку для шифра *Trivium 64* с помощью разработанного алгоритма.

Декомпозиционное множество, описанное в статьях в [14] и [16], и значение оценочной функции, подсчитанное на используемом в настоящей работе сервере, приведены на рисунке 11. Результат, полученный в ходе исследований, представлен на рисунке 12.

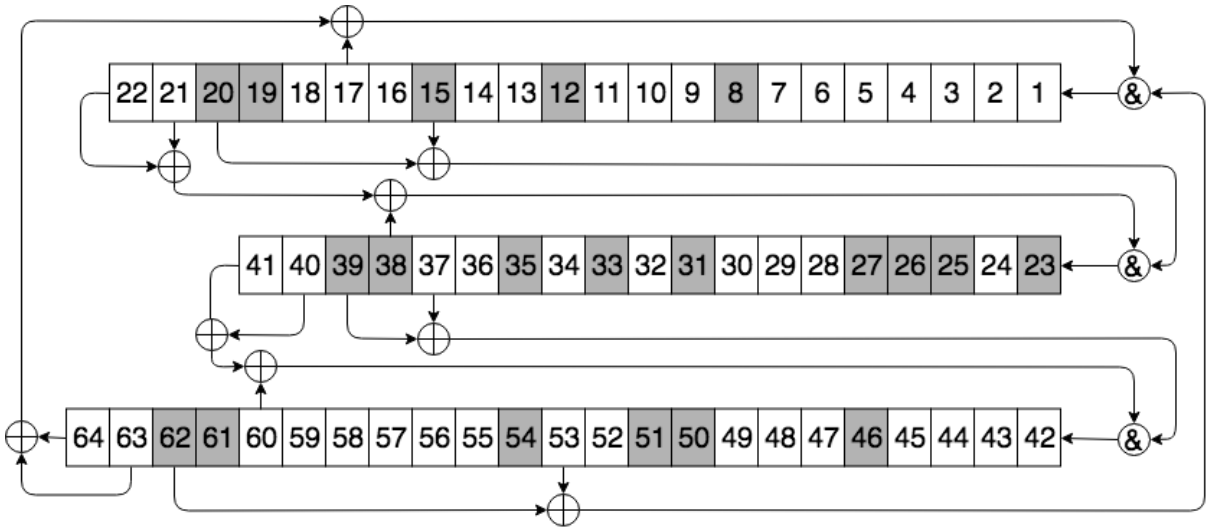

Рисунок 11. Декомпозиционное множество для шифра *Trivium 64* из статей [14] и [16].

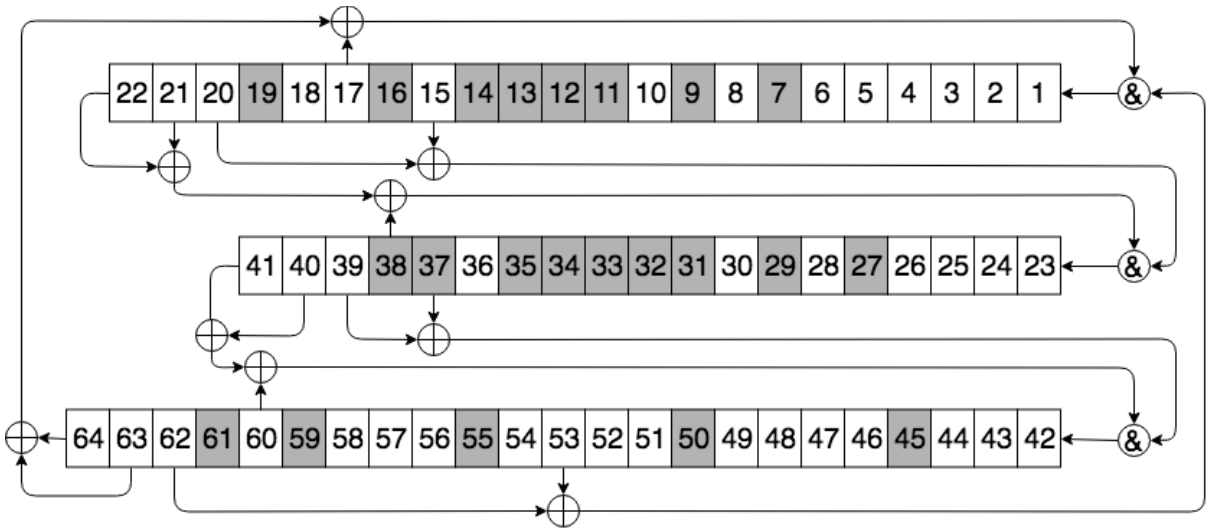

Рисунок 12. Декомпозиционное множество для шифра *Trivium 64*, полученное в настоящей работе.

Помимо этого было произведено сравнение алгоритмов, основанных на метаэвристических схемах *поиск с запретами* и *эволюционный алгоритм*, с использованием одинаковых вычислительных мощностей, а именно серверные мощности Кафедры компьютерных технологий Университета ИТМО. Для проведения такого сравнения были определены равные условия. Каждому алгоритму был выдан некоторый лимит на вызов оценочной функции. Алгоритм завершался при исчерпании выделенного ему лимита и производилось сравнение полученных результатов. Были определены следующие значения лимитов на вызов оценочной функции: 10, 50, 100, 200, 500, 1000. Результаты данного эксперимента приведены в таблице 7. Их визуализация представлена на рисунке 13.

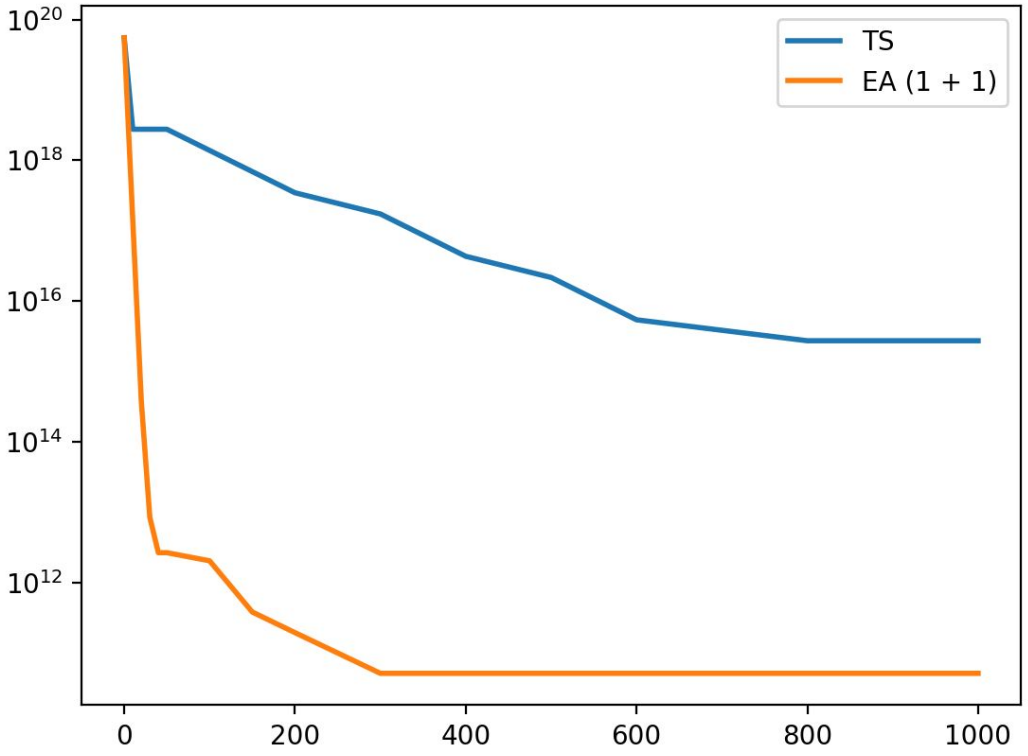

Рисунок 13. Зависимость числа вызовов оценочной функции от минимального полученного значения

Таблица 7. Результаты сравнения алгоритмов

Число вызовов оценочной функции	ЭА (1 + 1)	Поиск с запретами
10	$1.49 \cdot 10^{17}$	$2.77 \cdot 10^{18}$
50	$2.64 \cdot 10^{12}$	$2.77 \cdot 10^{18}$
100	$1.92 \cdot 10^{12}$	$1.38 \cdot 10^{18}$
200	$1.52 \cdot 10^{11}$	$3.46 \cdot 10^{17}$
500	$3.88 \cdot 10^{10}$	$1.08 \cdot 10^{16}$
1000	$3.54 \cdot 10^{10}$	$5.40 \cdot 10^{15}$

Также во время проведения экспериментов было получено достаточное число декомпозиционных множеств, построенных с использованием реализованной программной среды, которые алгоритм не смог улучшить за отведенный ему лимит на использование ресурсов. Для них была проведена оценка трудоемкости на больших выборках с целью увеличения точности полученного значения. И лишь для одного шифра удалось получить декомпозиционное множество с лучшей оценкой трудоемкости по сравнению с множеством, описанным в статьях [14] и [16].

Выводы по главе 3

В данной главе были проведены вычислительные эксперименты по построению guess-and-determine атак на следующие шифры: *A5/1*, *Trivium 64*, *Bivium*. Для каждого из перечисленных генераторов ключевого потока была определена эмпирическая зависимость граничного значения оценочной функции от объема выборки. Для шифра *Trivium 64* был найдено минимальное значение объема выборки, при которых получаемые

прогнозы оказывались относительно точными (не ухудшались с увеличением объема выборки).

Была построена новая *guess-and-determine* атака на шифр *Trivium 64*, оценка трудоемкости которой меньше чем у авторов оценочной функции, используемой в данной работе. Также было произведено сравнение метаэвристических схем *поиск с запретами* и *эволюционного алгоритма*.

Заключение

Была успешно реализована программная среда для построения декомпозиционных представлений трудных *SAT*-задач, кодирующих проблемы криптоанализа.

Была предложена стратегия адаптивного изменения объема выборки, которая не применялась ранее для построения *guess-and-determine* атак на криптографические алгоритмы. По результатам экспериментов было показано, что новая стратегия существенно ускоряет алгоритм на первых итерациях, на которых происходит спуск до некоторого граничного значения оценочной функции. Также данная стратегия может использоваться для определения объема выборки, достаточного для относительно точного прогноза времени реализации соответствующей *guess-and-determine* атаки.

С применением разработанных методов были построены guess-and-determine атаки на следующие генераторы ключевого потока: *A5/1*, *Trivium 64*, *Bivium*. Для шифра *Trivium 64* было найдено декомпозиционное множество с меньшей оценкой трудоемкости, чем описанное в статьях [14] и [16]

В ближайшем будущем предполагается продолжить начатые исследования по следующим направлениям:

1) Рассмотреть потоковые шифры *E0*, *Grain*, *Trivium*.
2) Разработать методы подбора оптимальных параметров для *SAT*-решателей.
3) Исследовать *переменные Цейтина* на предмет возможности их включения в декомпозионное множество.
4) Применить разработанные алгоритмы к криптоанализу блочных шифров.

Список источников

1. Cryptol documentation. https://cryptol.net/documentation.html
2. *Janicic P.* URSA: a system for uniform reduction to SAT // Logical Methods in Computer Science. – 2012. – Vol. 8. – №. 3. – P. 1-39.
3. Отпущенников И. В., Семенов А. А. Технология трансляции комбинаторных проблем в SAT // Прикладная дискретная математика. – 2011. – Т. 11. – №. 1. – С. 96-115.
4. *Otpuschennikov I., Semenov A., Gribanova I., Zaikin O., Kochemazov S.* Encoding Cryptographic Functions to SAT Using Transalg System // ECAI conference. – 2016.
5. Minisat: SAT-solver. – URL: https://github.com/niklasso/minisat
6. SAT Competition. – URL: http://www.satcompetition.org
7. Lingeling: SAT-solver. – URL: http://fmv.jku.at/lingeling/
8. Configurable SAT Solvers Challenge 2014. – URL: http://aclib.net/cssc2014/
9. Heule M., Kullmann O., Wieringa S., Biere A. Cube and conquer: Guiding CDCL SAT solvers by lookaheads // Hardware and Software: Verification and Testing: 7th International Haifa Verification Conference, HVC 2011, Haifa, Israel, December 6-8, 2011, Revised Selected Papers. – Springer, 2012. – Vol. 7261. – P. 50.
10. ROKK: SAT-solver. – URL: http://satcompetition.org/edacc/sc14/experiment/20/solver-configurations/1472
11. SATELlite: CNF minimizer. – URL: http://minisat.se/SatELite.html
12. eSTREAM: the ECRYPT Stream Cipher Project. – URL: http://www.ecrypt.eu.org/stream/
13. Castro Lechtaler A., Cipriano M., García E., Liporace J., Maiorano A., Malvacio E. Model design for a reduced variant of a Trivium Type Stream

Cipher // Journal of Computer Science and Technology. – 2014. – Vol. 14. – №. 1. – P. 55-58.

14. *Semenov A., Zaikin O.* Algorithm for Finding Partitionings of Hard Variants of Boolean Satisfiability Problem with Application to Inversion of Some Cryptographic Functions // SpringerPlus. – 2016. – Vol. 5. – P. 554-554.
15. *Metropolis N., Ulam S.* The Monte Carlo Method // Journal of the American Statistical Association. – 1949. – Vol. 44. – №. 247. – P. 335-341.
16. *Semenov A., Zaikin O., Otpuschennikov I., Kochemazov S., Ignatiev A.* On cryptographic attacks using backdoors for SAT // The Thirty-Second AAAI Conference on Artificial Intelligence. – IEEE, 2018. – P. 6641-6648.
17. Иркутский суперкомпьютерный центр. – URL: http://hpc.icc.ru/