Министерство науки и высшего образования
Российской Федерации

**Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)**

Факультет *«Фундаментальные науки»*

Кафедра *«Математическое моделирование»*

Расчётно-пояснительная записка

к курсовой работе

на тему

Построение и анализ криптосистемы, основанной на передаче информации по открытому каналу связи с использованием композиции проблем дискретного логарифмирования и сопряжённости в группах с условиями C(6)-T(3)

по дисциплине

«Численные методы решения задач теории управления»

Студент группы ФН12-61 _____ *Д.А. Касиянчук*
(подпись, дата)

Руководитель курсовой работы _____ *Н.В. Безверхний*
(подпись, дата)

Москва — 2021

Министерство науки и высшего образования
Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования «Московский государственный технический университет имени Н. Э. Баумана (национальный исследовательский университет)» (МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ
Заведующий кафедрой _____

_____ _____

«____» _____ 20___ г.

ЗАДАНИЕ
на выполнение курсовой работы

по дисциплине
«Численные методы решения задач теории управления»

Студент группы ФН12-61
Касиянчук Дмитрий Алексеевич
(Фамилия, Имя, Отчество)

Тема курсовой работы: **Построение и анализ криптосистемы, основанной на передаче информации по открытому каналу связи с использованием композиции проблем дискретного логарифмирования и сопряжённости в группах с условиями C(6)-T(3)**

Направленность КР исследовательская
(учебная, исследовательская, практическая, производственная и др.)

Источник тематики _____

График выполнения работы: 25% к __ нед., 50% к __ нед., 75% к __ нед., 100% к __ нед.

Задание _____

Оформление курсовой работы:

Расчётно-пояснительная записка на ____ листах формата А4.

Перечень графического (иллюстративного) материала _____

Дата выдачи задания «____» _____ 20___ г.

Руководитель курсовой работы _____ *Н.В. Безверхний*
(подпись, дата)

Студент _____ *Д.А. Касиянчук*
(подпись, дата)

Примечание. Задание оформляется в двух экземплярах: один выдается студенту, второй хранится на кафедре.

Содержание

1. Введение

Целью работы является изучение понятия односторонней функции и схемы формирования секретного ключа при обмене информацией по открытому каналу связи. При формировании односторонней функции предполагается использовать алгоритмы решения проблем дискретного логарифмирования и сопряжённости в группах с условиями C(6)-T(3).

Задачи работы:

1. Изучить комбинаторные методы представления групп.
2. Познакомиться с теорией малых сокращений.
3. Рассмотреть алгоритм решения проблем дискретного логарифмирования и сопряжённости в группах с условиями C(6)-T(3) и соответствующую схему формирования секретного ключа.
4. Сгенерировать группу с копредставлением рассматриваемого типа.
5. Смоделировать процесс формирования ключа абонентами A и B.
6. Вычислить ключ и оценить сложность прямой и обратной вычислительной процедур.

2. Теория малых сокращений

2.1. Основные понятия и определения

Пусть F - произвольное множество символов и для каждого символа $f \in F$ определён обратный символ $f^{-1} \in F^{-1}$, при этом $(f^{-1})^{-1} = f$.

Определение 1 *Множество $X = F \cup F^{-1}$ называется групповым алфавитом, а элементы этого множества - буквами.*

Определение 2 *Словом w в групповом алфавите X называется конечная строка букв $w = x_1 \dots x_n$, $x_i \in X$.*

Определение 3 *Словом w,обратным к слову $w = x_1 \dots x_n$, называется слово $w = x_n^{-1} \dots x_1^{-1}$.*

Определение 4 *Длиной элемента $w = x_1 \dots x_n$ называется целое число n и обозначается через $|w|$.*

Определение 5 *Слова w_1 и w_2 в символах $a, b, c...$ называют эквивалентными, если w_1 можно преобразовать в w_2 за конечное число шагов, каждый из которых состоит в выполнении одной из следующих операций:*

1. *Вставка одного из слов $A, A^{-1}, B, B^{-1}, ..$ или одного из слов, тривиально равных единице между любыми двумя соседними символами слова w_1 или перед словом w_1, или после него.*

2. *Вычёркивание из слова отрезка (части), совпадающего с одним из слов $A, A^{-1}, B, B^{-1}, ..$, или словом, тривиально равным единице.*

Определение 6 *Если $P = Q$, то есть слово PQ^{-1} образует нейтральный группы G, то слово PQ^{-1} называют определяющим соотношением.*

Определение 7 *Выражение $< a, b, ...|A, B, .. >$, где a,b,... - образующие, а A,B,.. - соотношения группы G относительно соответствующего группового алфавита, называется копредставлением G.*

Определение 8 *Если каждый элемент F определяется некоторым словом в символах a, b, c, то символы a, b, c называются образующими символами группы F.*

Определение 9 *Каждый элемент из F , отличный от 1, единственным образом представим в виде несократимого слова $w = y_1 ... y_n$, в котором две последовательные буквы не образуют пар вида $y_k y_k^{-1}$.*

Определение 10 *Приведённое слово называют циклически несократимым, если y_n не является обратным к y_1.*

Определение 11 *Пустое слово и слова $aa^{-1}, bb^{-1}, ..$ называют тривиально равными единице или тривиальными словами.*

Определение 12 *Множество $R \in F$ называется симметризованным в том случае, если все элементы R циклически несократимы и для всех $r \in R$ циклические перестановки r и $r^{-1} \in R$.*

Определение 13 *Слово r называется куском относительно множества R, если существуют такие $w_1, w_2 \in R$, что $w_1 = ru_1$, $w_2 = ru_2$.*

Определение 14 *Картой называется конечный набор попарно непересекающихся вершин, рёбер и областей, удовлетворяющих следующим условиям:*

1. Если g - ребро из M, то существуют вершины m и n, такие что $\overline{g} = g \cup m \cup n$

2. Граница ∂C каждой области C из M связна, для некоторых рёбер $g_1, ..., g_n$ из M имеем $\partial C = \overline{g_1} \cup ... \cup \overline{g_n}$

Определение 15 *Диаграммой над группой F называется ориентированная карта M вмсете с функцией φ, сопоставляющей каждому ориентированному ребру g карты M метку $\varphi(g)$ из F таким образом, что если g - ориентированное ребро из M, а g^{-1} - противоположным образом ориентированное ребро, то $\varphi(g^{-1}) = \varphi(g)^{-1}$.*

Пусть α- путь в M, $\alpha = g_1...g_m$ то положим $\varphi\alpha = \varphi g_1...\varphi g_m$. Если D- область из M, то её меткой называется элемент $\varphi\alpha$, где α - граничный цикл области D.

Теорема 1 *(Лемма Ван Кампена) Слово $w = 1$ в группе $G = (X; R)$ тогда и только тогда, когда существует связная односвязная приведённая диаграмма M над копредставлением G, граничная метка которой $\varphi \equiv w$.*

3. Генерация копредставлений групп с условиями малого сокращения

В данном разделе рассмотен алгоритм генерации копредставлений групп с заданными условиями $C(p) - T(q)$. Для упрощения понимания работы алгоритма было решено реализовать его в объектно-ориентированном стиле. В качестве языка программирования был выбран Python 3.

В 3.1 - 3.3 представлены классы `Letter`, `Word` и `Corepresentation` для буквы, группового слова и копредставления группы соответственно, описаны основные методы и переменные данных классов. В 3.4 приведено описание разработанного алгоритма генерации копредставлений групп. В Приложении А представлен исходный код программы с описанием предложенных классов и методов на языке Python

3.1. Класс Letter

Объект класса `Letter` (далее - буква) хранит символ группового алфавита, а также показатель степени (1 или -1). Если показатель степени буквы равен -1, то данная буква является "обратной"по отношению к букве с аналогичным значением символа и противоположным значением степени.

В классе `Letter` определены операции сравнения объектов данного класса, получения обратной буквы и проверки на сократимость с другой буквой.

3.2. Класс Word

Для построения определяющих соотношений необходимо иметь возможность оперировать свойствами задающих их определяющих соотношений, т.е. словами в групповом алфавите, представленными классом `Word`. Слово представляет собой упорядоченный набор букв из группового алфавита. В качестве контейнера для хранения последовательности букв был выбран встроенный тип данных список (list).

Далее представлено описание основых методов, реализованных в классе `Word`. Все описанные методы при применении к исходному слову не меняют его.

3.2.1. Получение подслова и удаление подслова

При помощи метода `GetSubword` осуществяется получение подслова заданной длины из начала исходного слова. Метод `DeleteSubword`, наоборот, позволяет получить все исходное слово, из которого удалено подслово заданной длины.

3.2.2. Циклический сдвиг

Метод `CyclicPermutation` возвращает исходное слово, в котором произведена циклическая перестановка букв. Осуществляется циклический сдвиг вправо на k символов (по умолчанию - на 1). В случае, если k по модулю превосходит длину слова l, то осуществляется циклический сдвиг на $(k \mod l)$ элементов.

3.2.3. Вставка подслова

Метод `Insert` осуществляет вставку подслова на заданную позицию в исходном слове и возвращает полученное слово.

3.2.4. Получение обратного слова

Метод `Reversed` возвращает слово, обратное исходному. Сначала создается слово, в котором те же буквы следуют в противоположном порядке, а затем каждая из букв заменяется на обратную.

3.2.5. Сокращение, R-сокращение и R-удлинение слова

Введем понятия R-сокращения и R-удлинения слова:

Определение 16 *Будем считать, что в слове w имеется R-сокращение, если существует такой элемент $r \in R$, что:*

1. $r \equiv r_1 r_2$,
2. $w \equiv w_1 w_2 w_3$,
3. $r_1 \equiv w_2$,
4. *слово r_2 является пустым или куском*
5. *слова $r_2^{-1} w_3, w_1 r_2^{-1}$ несократимы в свободной группе*

Определение 17 *Будем считать, что в слове w есть R-удлинение, если существует такой элемент $r \in R$, что:*

1. $r \equiv r_1 r_2$,
2. $w \equiv w_1 w_2 w_3$,
3. $r_1 \equiv w_2$,
4. *слово r_1 является куском.*

Метод `Reduced` возвращает слово, в котором произведены все возможные сокращения. Данный метод удаляет из слова все возможные тривиальные подслова. Если передан параметр `cyclic=True`, то производится также сокращение первого и последнего элементов, если это возможно.

Метод `R_reduced` возвращает слово, в котором произведены все R-сокращения.

Метод `R_elongated` возвращает слово, в котором произведено R-удлинение на заданной позиции.

3.3. Класс Corepresentation

Класс `Corepresentation` представляет собой копредставление группы. Объект данного класса хранит множество определяющих соотношений `__def_relations`. Для предотвращения повреждения объектов изменение множества определяющих соотношений после создания объекта класса невозможно, но доступ к ним без права изменения можно получить при помощи функции `GetRelators`.

В конструкторе класса `Corepresentation` также осуществляется провека на сократимость слов. Перед добавлением слова в `__def_relations` в нем осуществляются

все сокращения. Если слово пустое, то оно не добавляется в множество определяющих соотношений. Если после добавления всех подходящих слов множество определяющих соотношений пустое, то возвращается исключение:

```
ValueError('Множество определяющих соотношений пусто')
```

3.3.1. Симметризация

Метод `Symmetrization` производит построение симметризованного множества определяющих соотношений, т.е. множества, содержащего все возможные циклические перестановки слов из множества определяющих соотношений, а также слова, обратные им.

Для улучшения эффективности алгоритма при первом вызове данного метода производится построение множества и сохранение его в поле класса `Corepresentation`. Затем при каждом вызове данного метода возвращается значение соответствующего поля. Для удобства применения симметризованное множество хранится в виде отсортированного списка.

3.3.2. Куски

Вычисление кусков необходимо для проверки условия $C(p)$. Для нахождения кусков необходимо попарно сравнить все подслова всех элементов симметризованного множества копредставления. В предложенной реализации построение множества кусков производится при помощи метода `Pieces`.

3.3.3. Условие C(p)

Определение 18 *Условие $C(p)$: Никакой элемент из R не является произведением менее, чем p кусков.*

Проверка условия $C(p)$, реализованная в функции `C`, состоит в поиске такого слова, которое можно представить в виде произведения менее чем p кусков. В предложенной реализации выполняется проверка в цикле для каждого слова из множества определяющих соотношений.

Проверка для слова выполняется рекурсивно: если начало слова совпадает с некоторым куском, то вызывается проверка для слова, образованного отделением этого куска. Если для какого-либо слова условие не выполнилось, то функция возвращает значение *False*. Если такое слово найдено не было, то будет возвращено значение *True*.

3.3.4. Условие T(q)

Определение 19 *Условие $T(q)$: Пусть $3 \leq n < q$ и $y_1, ..., y_n$- элементы из R, причём последовательные элементы $y_i y_{i+1}$ не являются взаимно обратными, тогда хотя бы одно из произведений $y_1 y_2, ..., y_{h-1} y_h, y_h y_1$ приведено.*

Проверка условия $T(q)$ является в общем случае требует перебора размещений без повторений от 3 до $q-1$ слов из симметризованного множества соотношений. В случае,

если найдено размещение, не удовлетворяющее данному условию, то возвращается *False*. Если такое размещение найдено не было, то будет возвращено *True*.

В рамках данного исследования требуется только проверка условия $T(3)$. Данное условие является тривиальным, т.е. ему удовлетворяет любое множество определяющих соотношений. Поэтому реализация проверки этого условия в рамках данного исследования не требуется.

3.4. Алгоритм генерации копредставлений с заданными условиями

Генерация копредставлений с заданными параметрами осуществляется при помощи функции-генератора `GenerateCorepresentation`. Сначала вызывается функция `GenerateRelators`, генерирующая множество слов, удовлетворяющих заданным требованиям (количество букв из группового алфавита, минимальная и максимальная длина слова), которые могут быть использованы для построения множества определяющих соотношений. Затем осуществляется перебор всех комбинаций из r элементов (r - требуемое количество слов в определяющем соотношении) без повторений с проверкой условий $C(p) - T(q)$ для соответствующего копредставления.

Если для некоторого копредставления данные условия были выполены, то функция-генератор возвращает найденное копредставление и приостанавливает работу. При следующем вызове перебор продолжается.

4. Применение копредставлений групп с условиями малого сокращения для формирования секретного ключа

В данной работе рассматривается возможность создания алгоритма получения ключей, основанного на проблеме слов в группах, копредставление которых удовлетворяет условиям $C(6) - T(3)$. Таким образом, задача заключается в разработке процедуры получения общего секретного ключа с использованием открытого канала связи.

4.1. Формирование секретного ключа с применением композиции проблем дискретного логарифмирования и сопряженности

Участники:

1. Абонент A;
2. Абонент B;
3. Противник

Открытая информация:

1. Группа $G = \langle X; R\rangle$, копредставление которой удовлетворяет условиям $C(6) - T(3)$;
2. Слова w и h, являющиеся элементами бесконечного порядка в G

Алгоритм получения секретного ключа:

1. Абонент A выбирает произвольные ненулевые числа $n_1, m_1 \in \mathbb{Z}\backslash\{0\}$ (закрытый ключ), вычисляет открытый ключ $K_1 = h^{n_1} w^{m_1} h^{-n_1}$ и отправляет его абоненту B по открытому каналу связи.
2. Абонент B выбирает произвольные ненулевые числа $n_2, m_2 \in \mathbb{Z}\backslash\{0\}$ (закрытый ключ), вычисляет открытый ключ $K_2 = h^{n_2} w^{m_2} h^{-n_2}$ и отправляет его абоненту A по открытому каналу связи.
3. Абонент A вычисляет секретный ключ K_A:

$$K_A = h^{n_1} K_2^{m_1} h^{-n_1} = \ldots$$
$$\ldots = h^{n_1} \cdot \underbrace{(h^{n_2} w^{m_2} h^{-n_2}) \cdot (h^{n_2} w^{m_2} h^{-n_2}) \cdot \ldots \cdot (h^{n_2} w^{m_2} h^{-n_2})}_{m_1 \text{ произведений}} \cdot h^{-n1} = \ldots$$
$$\ldots = h^{n_1+n_2} w^{m_1 m_2} h^{-(n_1+n_2)} \quad (4.1)$$

4. Абонент B вычисляет секретный ключ $K_B = h^{n_2} K_1^{m_2} h^{-n_2} = h^{n_1+n_2} w^{m_1 m_2} h^{-(n_1+n_2)}$ аналогично (4.1)
5. Очевидно, что $K_A = K_B = K$.

   Таким образом, получен общий секретный ключ K.

Задача противника - получить секретный ключ K. Для этого необходимо найти одну из пар закрытых ключей (m_1, n_1 или m_2, n_2).

Очевидно, что при передаче слова K_1 по открытому каналу, графически равному слову $h^{n_1}w^{m_1}h^{-n_1}$, определение пары закрытых ключей m_1, n_1 является сравнительно простой задачей. Поэтому требуется заменить $h^{n_1}w^{m_1}h^{-n_1}$ некоторым словом v, которое равно $h^{n_1}w^{m_1}h^{-n_1}$ в группе G.

4.2. Применение односторонней функции для получения открытого ключа

Для получения открытого ключа K_i была предложена следующая односторонняя функция $F(w, m, h, n)$. На вход $F(w, m, h, n)$ подаются слова w, h и пара закрытых ключей m_i, n_i. Функция $F(w, m, h, n)$ возвращает слово $v = h^n w^m h^{-n}$, в котором проводится некоторое количество преобразований:

1. вставка тривиального слова $x_i x_i^{-1}$, где $x_i \in X$ выбирается случайным образом
2. вставка слова из симметризованного множества определяющих соотношений
3. проведение R-удлинения
4. проведение свободных сокращений
5. проведение R-сокращений

Количество этих преобразований выбирается произвольно. Преобразование 1 применяется с вероятностью 20%, преобразования 2 и 3 - с вероятностью 30%, остальные - с вероятностью 10%. Преобразования 1-3 можно выполнить всегда, тогда как преобразования 4 и 5 при применении второй раз подряд не произведут изменений.

5. Оценка производительности алгоритма

Были произведены 2 серии тестов для выявления зависимости скорости роста времени выполнения алгоритма при изменении параметров односторонней функции.

Далее используются обозначения:

n - количество используемых букв из группового алфавита

l - длина слов из множества определяющих соотношений

r - количество слов в множестве определяющих соотношений

k - количество преобразований, примененных односторонней функцией.

Серия 1: (произведено 10 тестов)

$$n = 5,\ l = 5,\ r = 3$$

$$w = abcde, h = acada$$

Построим зависимость времени получения ключа от параметров n_1 и n_2. Зафиксируем при этом параматры $m_1 = 20, m_2 = 20$. Для уменьшения влияния составляющей времени проведения "маскирующих"преобразований зафиксируем $k = 0$

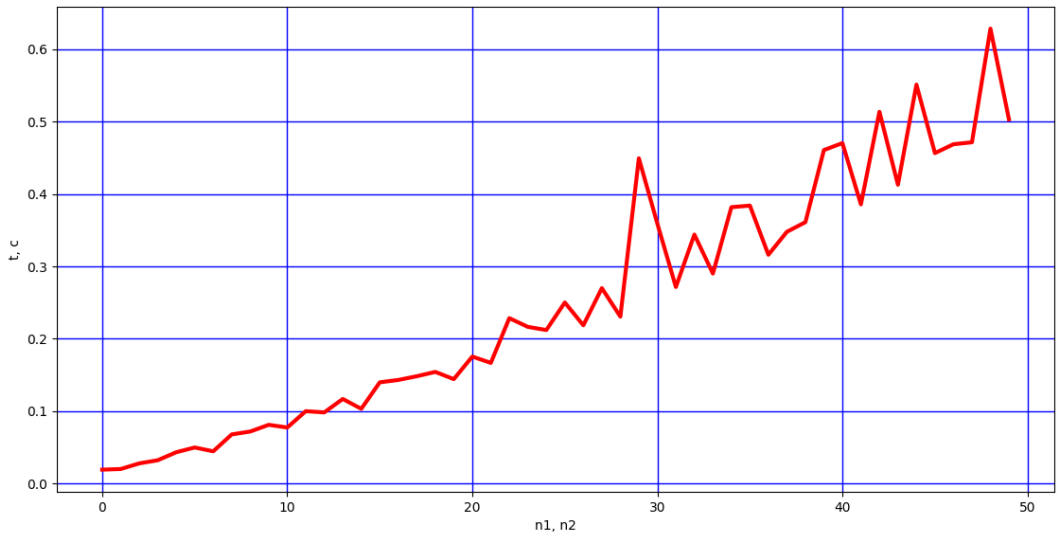
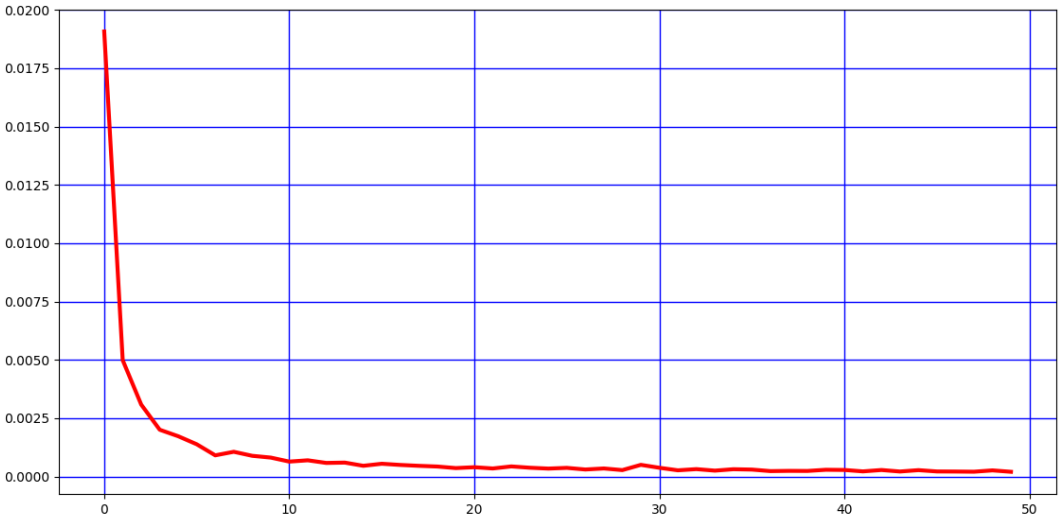


Рис. 1. Зависимость времени получения ключа от параметров n_1 и n_2

Проанализируем скорость роста полученной зависимости, построив график $\frac{t}{n^2}$:



Заметим, что при $n \longrightarrow \infty$ значение функции $\dfrac{t}{n^2}$ стремится к константе, что означает, что скорость роста зависимости $t(n_1, n_2)$ не более чем квадратичная.

Построим зависимость времени получения ключа от параметров m_1 и m_2. Зафиксируем при этом параматры $n_1 = 20,\ n_2 = 20,\ k = 0$.

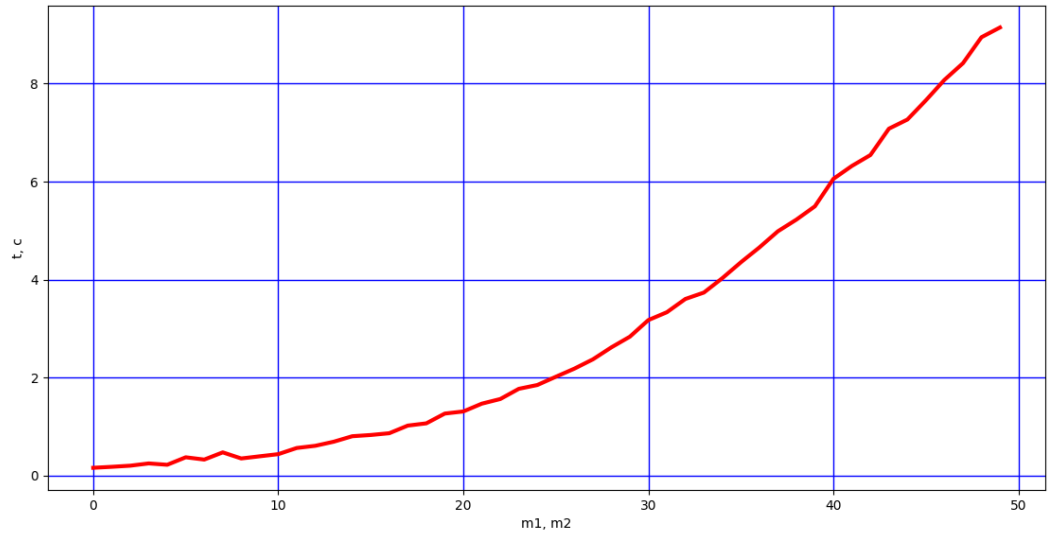
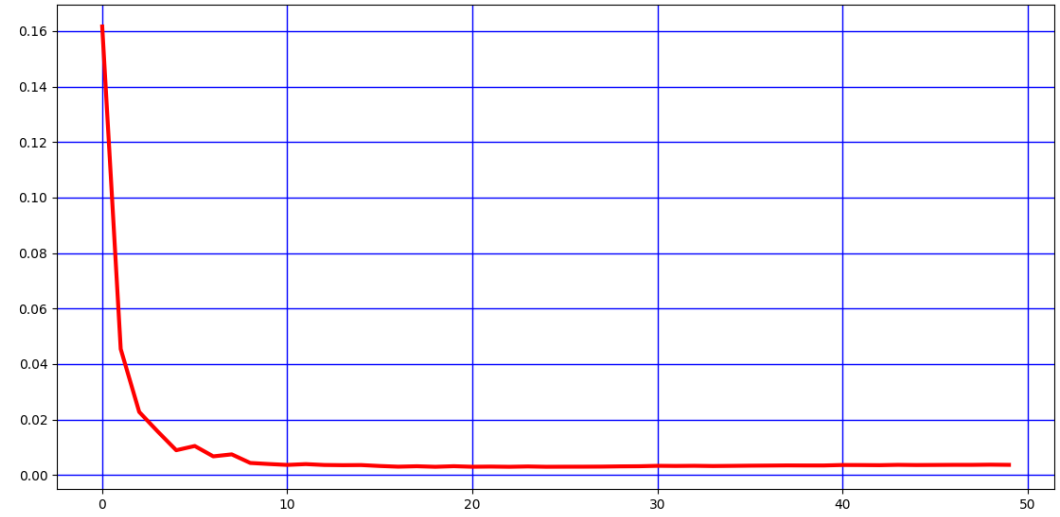


Рис. 2. Зависимость времени получения ключа от параметров m_1 и m_2

Проведя аналогичные рассуждения для $\dfrac{t}{m^2}$, получим, что скорость роста зависимости $t(m_1, m_2)$ близка к квадратичной.



Заметим, что зависимость $t(m_1, m_2)$ растет быстрее, чем $t(n_1, n_2)$, но скорость роста отличается только на константу.

Проанализируем скорость роста зависимости $t(k)$.

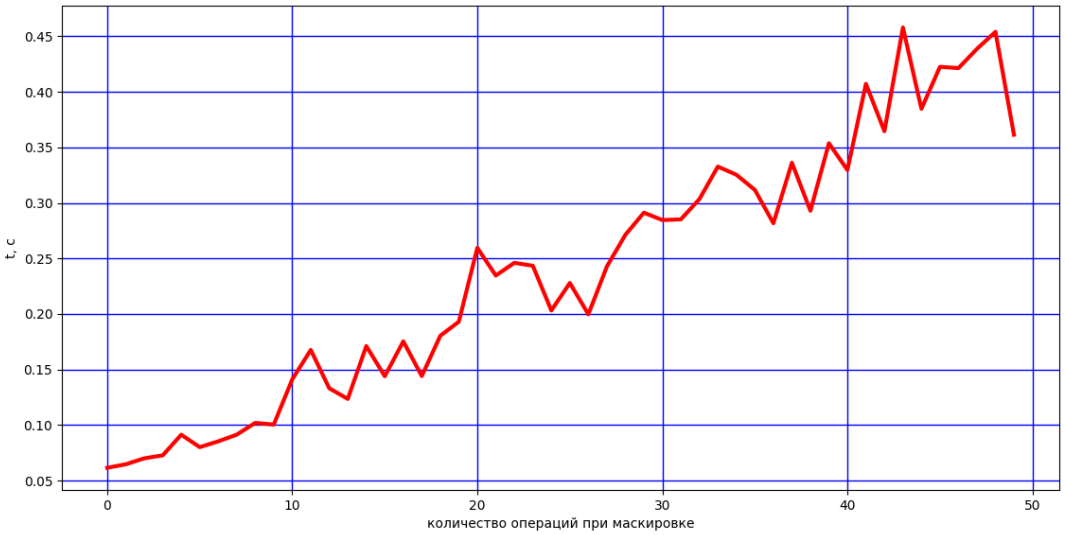


Рис. 3. Зависимость времени получения ключа от количества преобразований k, выполняемых односторонней функцией

Заметим, что данная зависимость является линейной. Это объясняется тем, что количество уже проведенных операций не оказывает влияния на время проведения новой операции. Также отметим, что время проведения различных преобразований в односторонней функции значительно отличается, поэтому даже при достаточно большом количестве тестов график имеет "пики".

Серия 2: (произведено 20 тестов)

$$n = 4,\ l = 3,\ r = 3$$

$$w = abcd,\ h = acad$$

Построенные копредставления:

```
<a, b, c, d;  a^(-1) d^(-1) c ,  d c^(-1) b ,  d^(-1) c a^(-1) >
<a, b, c, d;  a^(-1) d^(-1) c ,  d c^(-1) b ,  c a^(-1) d^(-1) >
<a, b, c, d;  a^(-1) d^(-1) c ,  d c^(-1) b ,  b d c^(-1) >
<a, b, c, d;  a^(-1) d^(-1) c ,  d c^(-1) b ,  d^(-1) b^(-1) c >
<a, b, c, d;  a^(-1) d^(-1) c ,  d c^(-1) b ,  c d^(-1) b^(-1) >
<a, b, c, d;  a^(-1) d^(-1) c ,  d c^(-1) b ,  d a c^(-1) >
<a, b, c, d;  a^(-1) d^(-1) c ,  b^(-1) c d^(-1) ,  d c^(-1) b >
<a, b, c, d;  a^(-1) d^(-1) c ,  a c^(-1) d ,  d c^(-1) b >
<a, b, c, d;  a^(-1) d^(-1) c ,  d c^(-1) b ,  c^(-1) b d >
<a, b, c, d;  a^(-1) d^(-1) c ,  d c^(-1) b ,  c^(-1) d a >
<a, b, c, d;  a^(-1) d^(-1) c ,  d^(-1) c a^(-1) ,  d b a >
<a, b, c, d;  a^(-1) d^(-1) c ,  d b a ,  c a^(-1) d^(-1) >
<a, b, c, d;  a^(-1) d^(-1) c ,  d b a ,  b a d >
<a, b, c, d;  a^(-1) b^(-1) d^(-1) ,  a^(-1) d^(-1) c ,  d b a >
<a, b, c, d;  a^(-1) d^(-1) c ,  d a c^(-1) ,  d b a >
<a, b, c, d;  b^(-1) d^(-1) a^(-1) ,  a^(-1) d^(-1) c ,  d b a >
<a, b, c, d;  a^(-1) d^(-1) c ,  d^(-1) a^(-1) b^(-1) ,  d b a >
<a, b, c, d;  a^(-1) d^(-1) c ,  a c^(-1) d ,  d b a >
<a, b, c, d;  a^(-1) d^(-1) c ,  c^(-1) d a ,  d b a >
<a, b, c, d;  a^(-1) d^(-1) c ,  a d b ,  d b a >
```

Представленные ниже результаты подтверждают предположения, сделанные на основе первой серии тестов

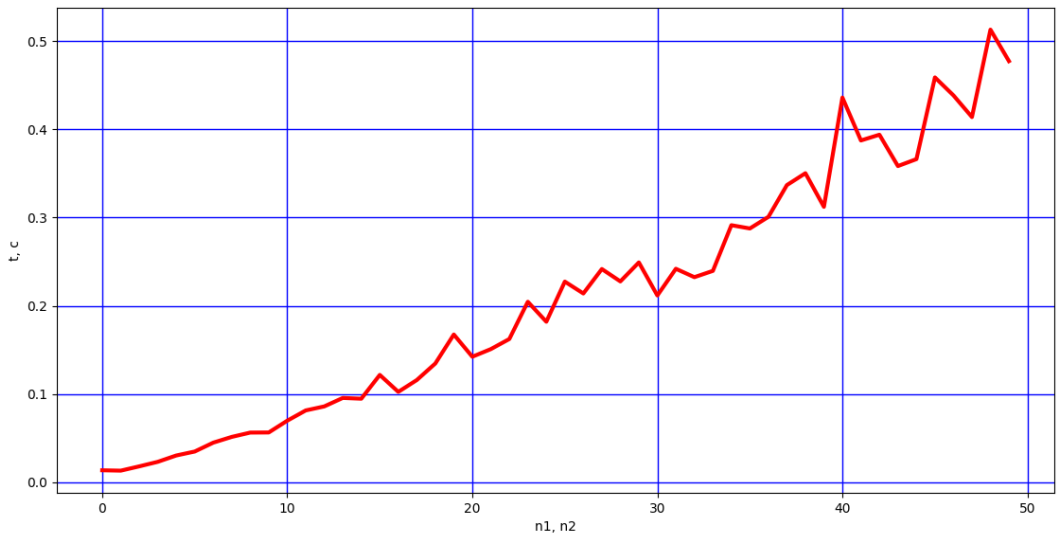


Рис. 4. Зависимость времени получения ключа от параметров n_1 и n_2

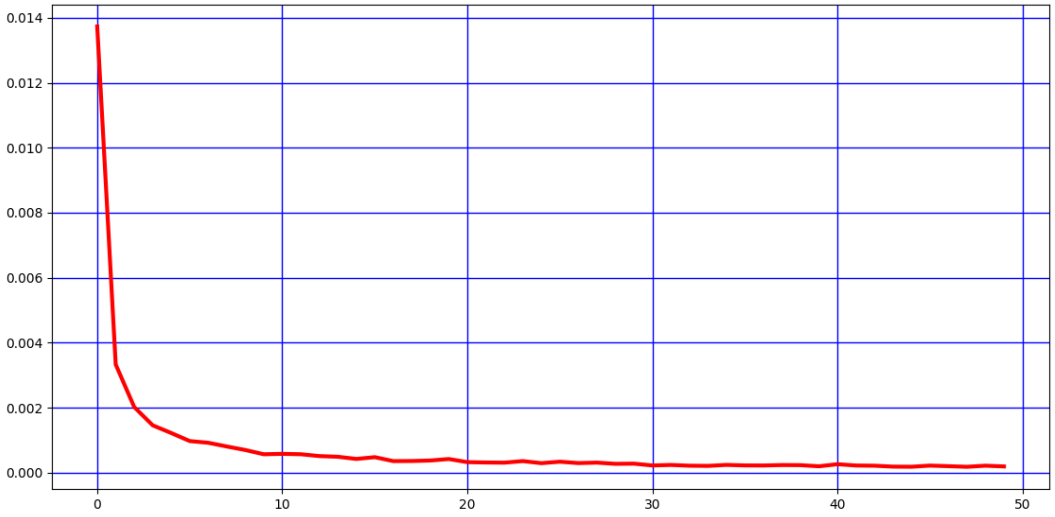


Рис. 5. Зависимость $\frac{t}{n^2}$ от параметров n_1 и n_2

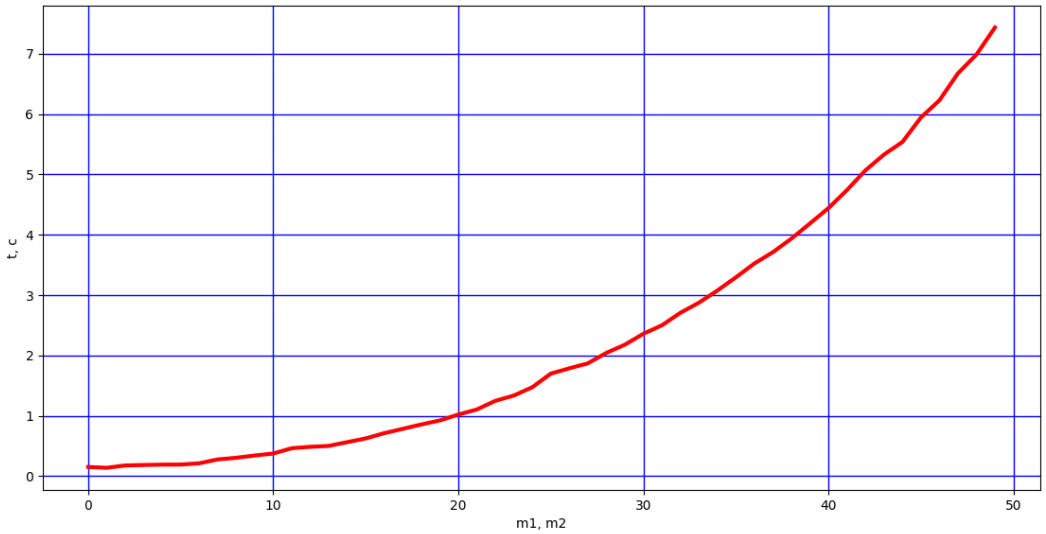


Рис. 6. Зависимость времени получения ключа от параметров m_1 и m_2

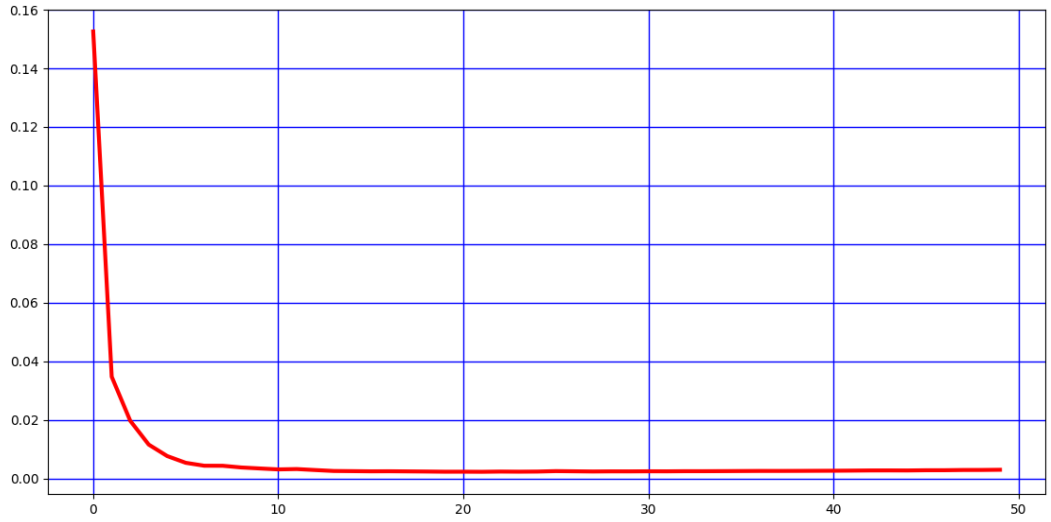


Рис. 7. Зависимость $\frac{t}{m^2}$ от параметров m_1 и m_2

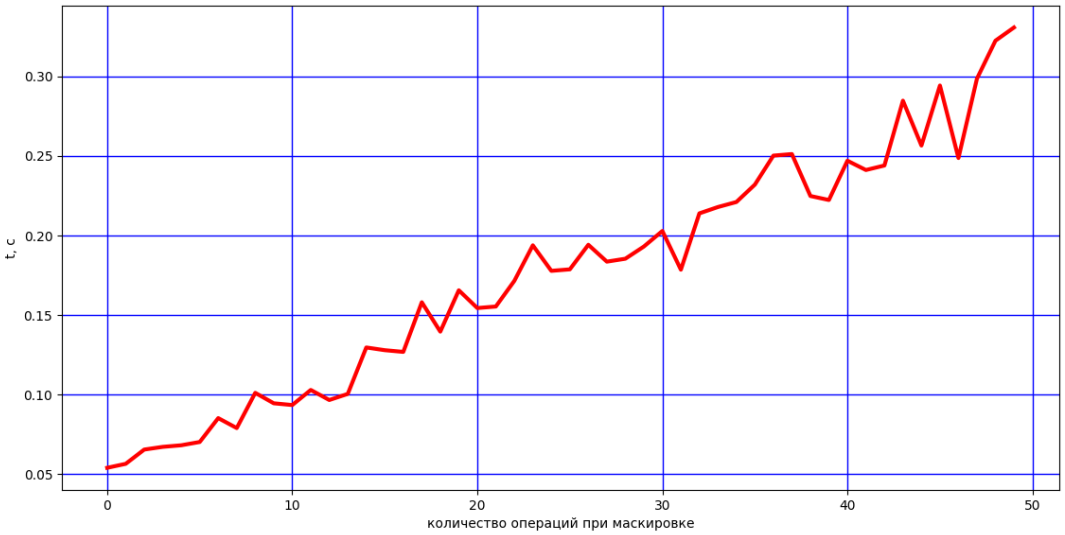


Рис. 8. Зависимость времени получения ключа от количества преобразований k, выполняемых односторонней функцией

6. Заключение

В ходе работы были изучены основные понятия и комбинаторные методы представления групп.Был разработан алгоритм, генерирующий копредставления с условиями малых сокращений $C(6) - T(3)$ и смоделирован процесс формирования ключа абонентами и с применением композиции проблем дискретного логарифмирования и сопряженности.

Было проведено исследование влияния различных параметров односторонней функции на время генерации секретного ключа.

Список литературы

1. Линдон Р., Шупп П. Комбинаторная теория групп. Пер. с англ. М.: Мир, 1980.

2. Глухов М.М. К анализу некоторых систем открытого распределения ключей, основанных на неабелевых группах. Математические вопросы криптографии. 2010. Т. 1 № 4, с. 5-22.

3. Безверхний Н.В., Чернышёва О.А. «Односторонние функции, основанные на проблеме дискретного логарифмирования в группах с условиями C(3)-T(6)», Наука и образование, 2014, № 10, с. 70-101.

Приложение А

Текст программы на языке Python:
Файл `letter.py`:

```
class Letter:

def __init__(self, sym, sign=1):
    self.__sym = sym
    self.__sign = sign

def Copy(self, other):
    self.__sym = other.__sym
    self.__sign = other.__sing
    return self

def __eq__(self, other):
    return self.__sym == other.__sym and self.__sign == other.__sign

def __ne__(self, other):
    return not (self == other)

def __lt__(self, other):
    if self.__sym == other.__sym:
        return (not self.IsInverse()) and other.IsInverse()
    return self.__sym < other.__sym

def __hash__(self):
    return hash((self.__sym, self.__sign))

def IsReductive(self, other):
    if self.__sym == other.__sym and self.__sign != other.__sign:
        return True
    return False

def IsInverse(self):
    return True if self.__sign < 0 else False

def Reverse(self):
    return Letter(self.__sym, -self.__sign)

def L2String(self):
    if self.__sign == 1:
        return "{0}".format(self.__sym)
    return "{0}^({1})".format(self.__sym, self.__sign)
```

Файл word.py:

```
import copy


class Word:

def __init__(self, let_list):
    if type(let_list) == Word:
        self.__letters = copy.deepcopy(let_list.__letters)
    else:
        self.__letters = copy.deepcopy(list(let_list))

def Copy(self):
    return Word(self.__letters)

def __eq__(self, other):
    if self.Size() != other.Size():
        return False
    for i in range(self.Size()):
        if self.__letters[i] != other.__letters[i]:
            return False
    return True

def __ne__(self, other):
    return not (self == other)

def __lt__(self, other):
    if self.Size() != other.Size():
        return self.Size() < other.Size()
    for i in range(self.Size()):
        if self.__letters[i] != other.__letters[i]:
            return self.__letters[i] < other.__letters[i]
    return False

def __mul__(self, other):
    return Word(self.__letters + other.__letters)

def __pow__(self, power):
    word_foo = self.Reverse() if power < 0 else self
    result = Word(word_foo)
    for i in range(abs(power) - 1):
        result = result * word_foo
    return result

def __hash__(self):
    return hash(tuple(self.__letters))
```

```
def __iter__(self):
    return iter(self.__letters)

def __next__(self):
    return next(self.__letters)

def __getitem__(self, key):
    return self.__letters[key]

def __setitem__(self, key, value):
    self.__letters[key] = value
    return self.__letters[key]

def Reverse(self):
    foo = Word(self)
    foo.__letters.reverse()
    for i in range(len(foo.__letters)):
        foo.__letters[i] = foo.__letters[i].Reverse()
    return foo

def IsSimple(self):
    for i in range(self.Size() - 1):
        if self.__letters[i].IsReductive(self.__letters[i + 1]):
            return False
    return True

def IsReductive(self):
    return not self.IsSimple()

def IsSimpleCyclically(self):
    if self.IsSimple() and
            not self.__letters[0].IsReductive(self.__letters[-1]):
        return True
    return False

def IsReductiveCyclically(self):
    return not self.IsSimpleCyclically()

def CyclicPermutation(self, k=1):
    foo = Word(self)
    for i in range(k % self.Size()):
        foo.__letters.insert(0, foo.__letters.pop())
    return foo

def Size(self):
    return len(self.__letters)
```

```
def IsEmpty(self):
    if self.Size() == 0:
        return True
    return False

def Reduced(self, cyclic=False):
    i = 0
    foo = Word(self)
    while i < foo.Size() - 1:
        if foo.__letters[i].IsReductive(foo.__letters[i + 1]):
            del foo.__letters[i:i + 2]
            if i > 0:
                i -= 1
        else:
            i += 1
    if cyclic:
        while foo.__letters[0].IsReductive(foo.__letters[-1]):
            foo.__letters = foo.__letters[1:-1]
    return foo

def W2String(self):
    word_str = ''
    for letter in self.__letters:
        word_str += letter.L2String() + ' '
    return word_str.strip()

def __str__(self):
    return self.W2String()

def print(self):
    print(self.W2String())

def GetSubword(self, subword_len):
    return Word(self.__letters[:subword_len])

def DeleteSubword(self, subword_len):
    return Word(self.__letters[subword_len:])

def IsSuperword(self, other):
    if other.Size() > self.Size():
        return False
    for i in range(other.Size()):
        if self.__letters[i] != other.__letters[i]:
            return False
    return True
```

```
def Insert(self, inserted_word, position):
    if position == 0:
        return Word(inserted_word.__letters + self.__letters)
    if position < self.Size():
        return Word(self.__letters[:position] + inserted_word.__letters +
             self.__letters[position:])
    return Word(self.__letters + inserted_word.__letters)

def R_reduced(self, corepresentation):
    word = Word(self)
    simplified = False
    r_min_size = corepresentation.Symmetrization()[0].Size()
    while not simplified:
        k = 0
        simplified = True
        while k < word.Size() - r_min_size + 1:
            for r in corepresentation.Symmetrization():
                if r.Size() >= word.Size() - k - 1:
                    continue
                for i in range(r.Size() - 1):
                    if r[i] != word[k + i]:
                        break
                else:
                    if r[-1] == word[k + r.Size() - 1]:
                        word = Word(word.__letters[:k] +
                            word.__letters[k + r.Size():])
                    else:
                        word = Word(word.__letters[:k] + [r[-1].Reverse()] +
                            word.__letters[k + r.Size() - 1:])
                    simplified = False
            word = word.Reduced()
            if simplified:
                break
            k += 1
    return word

def R_elongated(self, corepresentation, position):
    word = Word(self)
    for r in corepresentation.Symmetrization():
        if r.Size() > 1 and r[0] == word[position]:
            if position == word.Size() - 1:
                word = Word(word.__letters[:position]) * Word(r[1:]).Reverse()
            else:
                word = Word(word.__letters[:position]) * Word(r[1:]).Reverse() *
                    Word(word.__letters[position + 1:])
            break
    return word
```

Файл `alphabet_generator.py`:

```
from letter import Letter


def alphabet(n):
    for character in range(n):
        yield Letter("abcdefghijklmonpqrstuvwxyz"[character])
```

Файл corepresentation.py:

```
from word import Word
from alphabet_generator import alphabet
import itertools
from copy import deepcopy


class Corepresentation:

def __init__(self, word_list):
    self.__def_relations = set()
    self.__alphabet = set()
    for word in word_list:
        word_reduced = word.Reduced(cyclic=True)
        if word_reduced.Size() > 0:
            self.__def_relations.add(word_reduced)
            for letter in word_reduced:
                self.__alphabet.add(letter if not letter.IsInverse()
                        else letter.Reverse())
    if len(self.__def_relations) == 0:
        raise ValueError('Множество определяющих соотношений пусто')
    self.__symmetrization = None
    self.__pieces = self.Pieces()

def Symmetrization(self):
    if self.__symmetrization is not None:
        return self.__symmetrization
    result = set()
    for word in self.__def_relations:
        spam = Word(word)
        for i in range(word.Size()):
            result.add(spam)
            result.add(spam.Reverse())
            spam = spam.CyclicPermutation()
    self.__symmetrization = sorted(result)
    return self.__symmetrization

def Size(self):
    return len(self.__def_relations)

def Pieces(self):
    pieces = set()
    symmetrizated_relations = self.Symmetrization()
    for relation in symmetrizated_relations:
        for k in range(relation.Size()):
            subword = relation.GetSubword(k + 1)
```

```
            found = False
            for other_relation in symmetrizated_relations:
                if other_relation != relation and other_relation.IsSuperword(subwor
                    pieces.add(subword)
                    found = True
            if not found:
                break
    return pieces

def C(self, p):
    def C4Word(word, k):
        if word.Size() == 0:
            return False
        if k == 0:
            return True
        for piece in self.__pieces:
            if word.IsSuperword(piece):
                word_tail = word.DeleteSubword(piece.Size())
                if not C4Word(word_tail, k - 1):
                    return False
        return True

    for relation in self.__def_relations:
        if not C4Word(relation, p - 1):
            return False
    return True

def GetRelators(self):
    return list(deepcopy(self.__def_relations))


def GenerateRelators(n, l_min, l_max):
    letters = set()
    for _letter in alphabet(n):
        letters.add(_letter)
        letters.add(_letter.Reverse())
    result = set()
    for length in range(l_min, l_max + 1):
        for combination in itertools.combinations_with_replacement(letters, length)
            for replacement in itertools.permutations(combination):
                relator = Word(replacement)
                reduced_relator = relator.Reduced(cyclic=True)
                if reduced_relator.Size() >= l_min:
                    result.add(reduced_relator)
    return result
```

```
def GenerateCorepresentation(n, l, r, p):
    all_relations = GenerateRelators(n, l, l)
    for t in range(r, r + 1):
        for relations in itertools.combinations(all_relations, t):
            corepresentation = Corepresentation(relations)
            if corepresentation.C(p):
                yield corepresentation
```

Файл `key_generation.py`:

```
from alphabet_generator import alphabet
from corepresentation import *
from random import randint, choice


def CreateKey(w, m, h, n):
    return (h ** n) * (w ** m) * (h ** -n)


def Hide(word, n_operations, corepresentation, n_letters):
    letters = []
    for _letter in alphabet(n_letters):
        letters.append(_letter)
        letters.append(_letter.Reverse())
    word = Word(word)

    for i in range(n_operations):
        k = randint(0, 10)
        m = randint(0, word.Size() - 1)
        if k < 2:
            letter = choice(letters)
            trivial_word = Word([letter, letter.Reverse()])
            new_word = word.Insert(trivial_word, m)
        elif k < 5:
            relation = choice(corepresentation.Symmetrization())
            new_word = word.Insert(relation, m)
        elif k < 8:
            new_word = word.R_elongated(corepresentation, m)
        elif k < 9:
            new_word = word.Reduced()
        else:
            new_word = word.R_reduced(corepresentation)
        word = new_word
    return word
```

Файл `main.py`:

```
from letter import Letter
from key_generation import *
import timeit
import numpy as np
import matplotlib.pyplot as plt


l_a = Letter("a")
l_b = Letter("b")
l_c = Letter("c")
l_d = Letter("d")
l_e = Letter("e")


w_word = Word([l_a, l_b, l_c, l_d])
h_word = Word([l_a, l_c, l_a, l_d])

generated_corepresentations = []
gcr = GenerateCorepresentation(4, 3, 3, 6)
n_tests = 5
for i in range(n_tests):
    next_cr = next(gcr)
    def_relations = next_cr.GetRelators()
    print('<a, b, c, d; ', def_relations[0], ', ', def_relations[1], ', ', def_rela
    generated_corepresentations.append(next_cr)
"""
Генератор копредставлений
n - количество используемых букв из группового алфавита
l - длина слов из множества определяющих соотношений
r - количество слов в множестве определяющих соотношений
p - параметр условия C(p)
yield копредставление, удовлетворяющее условиям C(p)-T(q)
"""

m_A = 1
n_A = 20
m_B = 1
n_B = 20

n_operations_k1 = 20
n_letters_k1 = 3
n_operations_k2 = 20
n_letters_k2 = 3

n_A_diff = np.arange(0, 50)
```

```
n_B_diff = np.arange(0, 50)
time_plot = np.zeros(50, dtype=float)
time_plot_divx2 = np.zeros(50, dtype=float)
for i in range(50):
    start_time = timeit.default_timer()
    print(i)
    n_tests = len(generated_corepresentations)
    for cr in generated_corepresentations:
        K1 = CreateKey(w_word, m_A, h_word, n_A_diff[i])
        K2 = CreateKey(w_word, m_B, h_word, n_B_diff[i])

        K1_h = Hide(K1, n_operations_k1, cr, n_letters_k1)
        K2_h = Hide(K2, n_operations_k2, cr, n_letters_k2)
        K_A = CreateKey(K1_h, m_B, h_word, n_B_diff[i])
        K_B = CreateKey(K2_h, m_A, h_word, n_A_diff[i])

    time_plot[i] = (timeit.default_timer() - start_time) / n_tests
    time_plot_divx2[i] = time_plot[i] / (i + 1) ** 2

fig, ax = plt.subplots()
ax.plot(n_A_diff, time_plot, linewidth=3, color='r')
ax.set_xlabel('n1, n2')
ax.set_ylabel('t, c')
ax.grid(color='blue', linewidth=1)
plt.show()

_, ax = plt.subplots()
ax.plot(n_A_diff, time_plot_divx2, linewidth=3, color='r')
ax.grid(color='blue', linewidth=1)
plt.show()

m_A_diff = np.arange(0, 50)
m_B_diff = np.arange(0, 50)
time_plot = np.zeros(50, dtype=float)
for i in range(50):
    start_time = timeit.default_timer()
    print(i)
    n_tests = len(generated_corepresentations)
    for cr in generated_corepresentations:
        K1 = CreateKey(w_word, m_A_diff[i], h_word, n_A)
        K2 = CreateKey(w_word, m_B_diff[i], h_word, n_B)

        K1_h = Hide(K1, n_operations_k1, cr, n_letters_k1)
        K2_h = Hide(K2, n_operations_k2, cr, n_letters_k2)
        K_A = CreateKey(K1_h, m_B_diff[i], h_word, n_B)
        K_B = CreateKey(K2_h, m_A_diff[i], h_word, n_A)
```

```
    time_plot[i] = (timeit.default_timer() - start_time) / n_tests
    time_plot_divx2[i] = time_plot[i] / (i + 1) ** 2

_, ax = plt.subplots()
ax.plot(m_A_diff, time_plot, linewidth=3, color='r')
ax.set_xlabel('m1, m2')
ax.set_ylabel('t, c')
ax.grid(color='blue', linewidth=1)
plt.show()

_, ax = plt.subplots()
ax.plot(m_A_diff, time_plot_divx2, linewidth=3, color='r')
ax.grid(color='blue', linewidth=1)
plt.show()

n_operations = np.arange(50)
time_plot = np.zeros(50, dtype=float)
for i in range(50):
    start_time = timeit.default_timer()
    print(i)
    n_tests = len(generated_corepresentations)
    for cr in generated_corepresentations:
        K1 = CreateKey(w_word, m_A, h_word, n_A)
        K2 = CreateKey(w_word, m_B, h_word, n_B)

        K1_h = Hide(K1, n_operations[i], cr, n_letters_k1)
        K2_h = Hide(K2, n_operations[i], cr, n_letters_k2)
        K_A = CreateKey(K1_h, m_B, h_word, n_B)
        K_B = CreateKey(K2_h, m_A, h_word, n_A)

    time_plot[i] = (timeit.default_timer() - start_time) / n_tests

_, ax = plt.subplots()
ax.plot(n_operations, time_plot, linewidth=3, color='r')
ax.grid(color='blue', linewidth=1)
ax.set_xlabel('количество операций при маскировке')
ax.set_ylabel('t, c')
plt.show()
```