



МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего
образования

«МИРЭА – Российский технологический университет»
РТУ МИРЭА

Институт искусственного интеллекта
Базовая кафедра 252

ДОЛГОСРОЧНАЯ РАБОТА

по дисциплине

«Криптографические методы защиты информации»

Тема долгосрочной работы

«Метод встречи посередине»

Студент: Никишина Анна Александровна

Группа: ККСО-03-19

Руководитель
практической работы: Бондакова Ольга Сергеевна

Москва
2023

Содержание

1	Введение	3
1.1	Вводная часть	3
1.2	Описательная часть	3
2	Основная часть	4
3	Заключение	9
4	Приложения	10
4.1	Полная версия кода	10

1 Введение

1.1 Вводная часть

В настоящее время, защита информации является одной из ключевых задач, стоящих перед современным обществом. Вместе с развитием технологий и распространением цифровых средств обмена данными, возникает все больше угроз, связанных с конфиденциальностью и целостностью информации. Криптография, как наука о методах обеспечения конфиденциальности и целостности данных, играет важную роль в области защиты информации.

Одним из методов криптографической защиты данных является метод встречи посередине (англ. "meet-in-the-middle"). Этот метод основан на комбинировании двух шифрований, использующих различные ключи. Его особенностью является возможность сократить пространство поиска ключей, что увеличивает эффективность атак и позволяет обойти классические методы криптоанализа.

В рамках данной долгосрочной работы я поставила перед собой цель реализовать метод встречи посередине на языке программирования C++. Я стремлюсь создать программу, способную шифровать и дешифровать данные с использованием данного метода, а также провести анализ его эффективности и безопасности. Эта работа позволит нам глубже понять принципы криптографической защиты информации и применить их на практике.

В данном отчете мы представим результаты нашей работы, включая описание реализованного метода встречи посередине, процесс разработки программы, анализ полученных данных и обсуждение возможных применений и ограничений данного метода.

1.2 Описательная часть

Для реализации метода встречи посередине на языке C++, мы выбрали использовать 4-х раундовый XSP-шифр. XSP-шифр состоит из трех основных операций: X-OR (X), побитовой перестановки (S) и транспонирования (P). Каждый раунд шифрования включает эти операции для обеспечения безопасности данных.

Моя реализация XSP-шифра использует 64-битный ключ, который разделяется на 4 раундовых ключа по 16 бит для шифрования 16-битного блока данных. Это позволяет нам достичь более высокого уровня безопасности и обеспечить эффективность алгоритма.

Теперь давайте рассмотрим, что такое метод встречи посередине (meet-in-the-middle) и как он работает. Метод встречи посередине является методом атаки на криптографические системы, основанный на комбинировании двух шифрований, использующих разные ключи.

В нашей работе мы имеем 1000 пар блоков открытого текста (ОТ) и соответствующих шифротекстов (ШТ). На первой фазе метода встречи посередине перебираются все возможные комбинации ключей k_1 и k_2 . Для каждой комбинации генерируются полу-шифротексты (e_OT), полученные после прохождения двух раундов шифрования с использованием k_1 и k_2 соответственно. Затем пару k_1 и k_2 добавляют в массив списков (`list_all_keys`) по номеру e_OT .

На второй фазе метода встречи посередине перебираются все возможные комбинации ключей k_3 и k_4 . Для каждой комбинации генерируются полу-расшифрованные тексты (d_SHT). Если элемент в списке `list_all_keys[d_SHT]` не пустой, то перебираются все его элементы, и блоки k_1 , k_2 , k_3 , k_4 добавляются в массив `list_valid_keys`.

Для остальных блоков данных перебираются все элементы массива `list_valid_keys`. Вычисляются e_OT и d_SHT и проверяется их равенство. Если e_OT и d_SHT не совпадают, то элемент (k_1 , k_2 , k_3 , k_4) удаляется из списка.

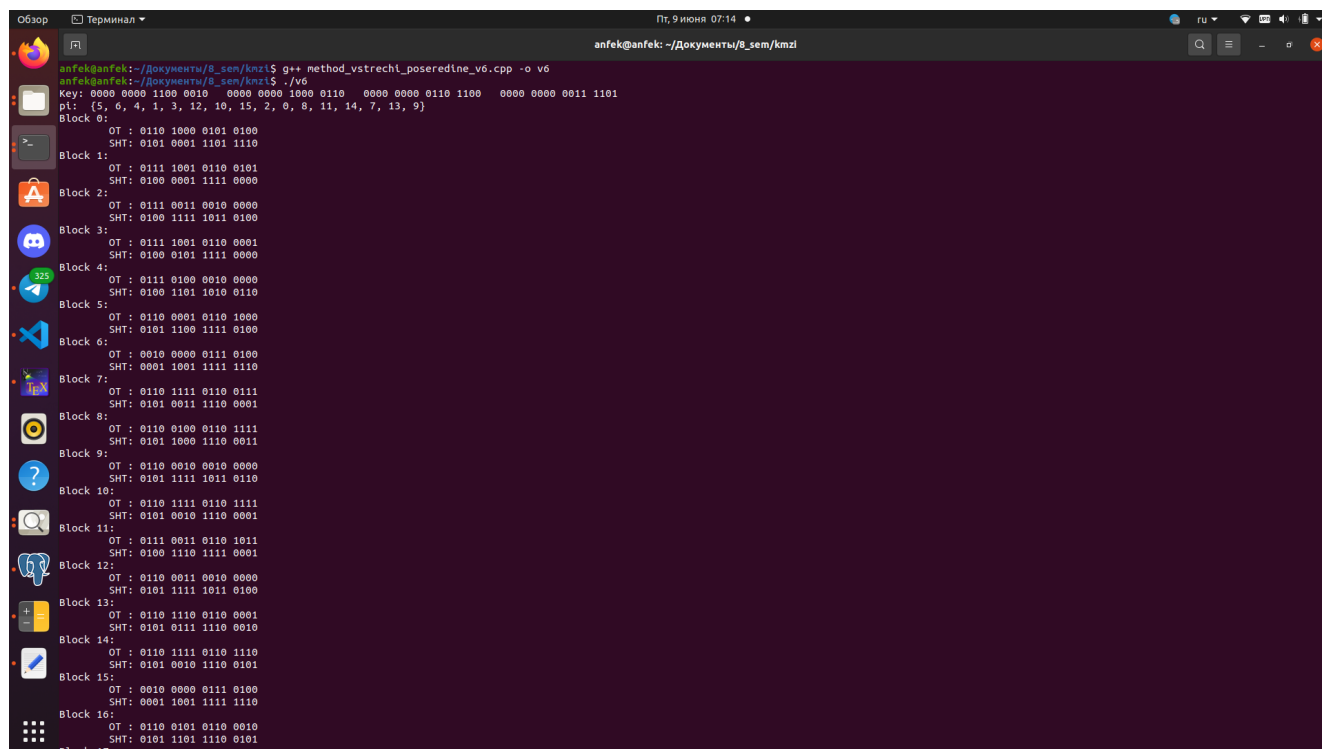
Таким образом, метод встречи посередине позволяет нам эффективно искать совпадения в парах блоков данных, используя комбинации ключей и массивы списков. Этот метод является эффективным инструментом для анализа безопасности криптографических систем и может быть применен в различных сферах, требующих защиты информации.

2 Основная часть

Моя программа содержит 3 класса:

- Класс Crypto – он содержит все методы и функции для работы с криптографией в нашей программе.
- Класс Gen_Text_Block – он берёт 1000 блоков текстов по 16 бит (2 байта) из файла и создаёт пары ОТ-ШТ.
- Класс Method_work – в нём непосредственно реализован алгоритм подбора ключа «Метод встречи посередине».

Для наглядности и ускорения работы ограничим наши ключи 8-ю битами и параллельно выводу на экран будем записывать промежуточные этапы работы программы в файл.



```
anfek@anfek:~/Документы/8_sem/kmz1$ g++ method_vstrechi_poseredine_v6.cpp -o v6
anfek@anfek:~/Документы/8_sem/kmz1$ ./v6
Key: 0000 0000 1100 0010 0000 0000 1000 0110 0000 0000 0110 1100 0000 0000 0011 1101
pi: {5, 6, 4, 1, 3, 12, 10, 15, 2, 0, 8, 11, 14, 7, 13, 9}
Block 0:
  OT : 0110 1000 0101 0100
  SHT: 0101 0001 1101 1110
Block 1:
  OT : 0111 1001 0110 0101
  SHT: 0100 0001 1111 0000
Block 2:
  OT : 0111 0011 0010 0000
  SHT: 0100 1111 1011 0100
Block 3:
  OT : 0111 1001 0110 0001
  SHT: 0100 0101 1111 0000
Block 4:
  OT : 0111 0100 0010 0000
  SHT: 0100 1101 1010 0110
Block 5:
  OT : 0110 0001 0110 1000
  SHT: 0101 1100 1111 0100
Block 6:
  OT : 0010 0000 0111 0100
  SHT: 0001 1001 1111 1110
Block 7:
  OT : 0110 1111 0110 0111
  SHT: 0101 0011 1110 0001
Block 8:
  OT : 0110 0100 0110 1111
  SHT: 0101 1000 1110 0011
Block 9:
  OT : 0110 0010 0010 0000
  SHT: 0101 1111 1011 0110
Block 10:
  OT : 0110 1111 0110 1111
  SHT: 0101 0010 1110 0001
Block 11:
  OT : 0111 0011 0110 1011
  SHT: 0100 1110 1111 0001
Block 12:
  OT : 0110 0011 0010 0000
  SHT: 0101 1111 1011 0100
Block 13:
  OT : 0110 1110 0110 0001
  SHT: 0101 0111 1110 0010
Block 14:
  OT : 0110 1111 0110 1110
  SHT: 0101 0010 1110 0101
Block 15:
  OT : 0010 0000 0111 0100
  SHT: 0001 1001 1111 1110
Block 16:
  OT : 0110 0101 0110 0010
  SHT: 0101 1101 1110 0101
Block 17:
```

Рис. 1: Компиляция и запуск программы.

После запуска программа выводит на экран истинное значение ключа Key, подстановку pi и 1000 пар блоков ОТ-ШТ.

Значение Key выводится на экран для того чтобы в последствии мы могли сравнить его с полученным, но в самой программе оно никуда из класса не передаётся.

```
Обзор Терминал
anfek@anfek: ~/Документы/8_sem/kmzi

Block 993:
  OT : 0110 0001 0111 0000
  SHT: 0101 1101 1111 1100

Block 994:
  OT : 0110 1110 0110 1001
  SHT: 0101 0110 1110 0010

Block 995:
  OT : 0110 1001 0111 0100
  SHT: 0101 0001 1111 1100

Block 996:
  OT : 0110 0111 0110 1110
  SHT: 0101 1010 1110 0101

Block 997:
  OT : 0010 1100 0111 0011
  SHT: 0001 0101 1110 1011

Block 998:
  OT : 0110 0100 0010 0000
  SHT: 0101 1101 1010 0110

Block 999:
  OT : 0111 0000 0110 0101
  SHT: 0100 1001 1111 0010

first_phase_1: 0000 0000 1111 1111 0000 0000 1111 1111 ff ff 255 1 255 255

first_phase_1 list all: 05536 10000
KEY first_phase_2: 0000 0000 0101 1111 0000 0000 0101 1000 0000 0000 1111 1111 0000 0000 1111 1111 5f 58 ff ff 91 8 95 88 255 25577

first_phase_2 list val: 131072 20000

other_recur: num = 1 list_valid_keys.size() = 131072 20000
other_recur: num = 2 list_valid_keys.size() = 131072 20000
other_recur: num = 3 list_valid_keys.size() = 131072 20000
other_recur: num = 4 list_valid_keys.size() = 131072 20000
other_recur: num = 5 list_valid_keys.size() = 131072 20000
other_recur: num = 6 list_valid_keys.size() = 131072 20000
other_recur: num = 7 list_valid_keys.size() = 131072 20000
other_recur: num = 8 list_valid_keys.size() = 131072 20000
other_recur: num = 9 list_valid_keys.size() = 131072 20000
other_recur: num = 10 list_valid_keys.size() = 131072 20000
other_recur: num = 11 list_valid_keys.size() = 131072 20000
other_recur: num = 12 list_valid_keys.size() = 131072 20000
other_recur: num = 13 list_valid_keys.size() = 131072 20000
other_recur: num = 14 list_valid_keys.size() = 131072 20000
other_recur: num = 15 list_valid_keys.size() = 131072 20000
other_recur: num = 16 list_valid_keys.size() = 131072 20000
other_recur: num = 17 list_valid_keys.size() = 131072 20000
other_recur: num = 18 list_valid_keys.size() = 131072 20000
other_recur: num = 19 list_valid_keys.size() = 131072 20000
other_recur: num = 20 list_valid_keys.size() = 131072 20000
other_recur: num = 21 list_valid_keys.size() = 131072 20000
other_recur: num = 22 list_valid_keys.size() = 131072 20000
other_recur: num = 23 list_valid_keys.size() = 131072 20000
other_recur: num = 24 list_valid_keys.size() = 131072 20000
other_recur: num = 25 list_valid_keys.size() = 131072 20000
other_recur: num = 26 list_valid_keys.size() = 131072 20000
other_recur: num = 27 list_valid_keys.size() = 131072 20000
```

Рис. 2: Работа фазы 1 и фазы 2.

Во время работы фазы 1 и 2 на экран выводятся текущие значения ключей, чтобы мы могли в реальном времени отслеживать прогресс выполнения программы. Затем по мере выполнения выводятся номера остальных пройденных блоков и текущее количество найденных ключей.

```
Обзор Терминал
anfek@anfek: ~/Документы/8_sem/kmzi

search_true_key: 0000 0000 0101 1000 0000 0000 0100 0100 0000 0000 1111 1111 0000 0000 1111 1001 58 44 ff f9 88 68 255 249
search_true_key: 0000 0000 0101 1100 0000 0000 0101 0100 0000 0000 1111 1111 0000 0000 1111 1001 5c 54 ff f9 92 84 255 249
search_true_key: 0000 0000 0101 1110 0000 0000 0101 0000 0000 0000 1111 1111 0000 0000 1111 1001 5e 50 ff f9 94 80 255 249
search_true_key: 0000 0000 0101 0001 0000 0000 0101 0110 0000 0000 1111 1111 0000 0000 1111 1010 51 56 ff fa 81 86 255 250
search_true_key: 0000 0000 0101 0011 0000 0000 0101 0010 0000 0000 1111 1111 0000 0000 1111 1010 53 52 ff fa 83 82 255 250
search_true_key: 0000 0000 0101 0101 0000 0000 0100 0110 0000 0000 1111 1111 0000 0000 1111 1010 55 46 ff fa 85 70 255 250
search_true_key: 0000 0000 0101 0111 0000 0000 0100 0010 0000 0000 1111 1111 0000 0000 1111 1010 57 42 ff fa 87 66 255 250
search_true_key: 0000 0000 0101 1001 0000 0000 0101 0100 0000 0000 1111 1111 0000 0000 1111 1010 59 54 ff fa 89 84 255 250
search_true_key: 0000 0000 0101 1011 0000 0000 0101 0000 0000 0000 1111 1111 0000 0000 1111 1010 5b 50 ff fa 91 80 255 250
search_true_key: 0000 0000 0101 1101 0000 0000 0100 0100 0000 0000 1111 1111 0000 0000 1111 1011 5f 50 ff fa 93 68 255 250
search_true_key: 0000 0000 0101 1111 0000 0000 0100 0000 0000 0000 1111 1111 0000 0000 1111 1010 5f 40 ff fa 95 04 255 250
search_true_key: 0000 0000 0101 0001 0000 0000 0100 0110 0000 0000 1111 1111 0000 0000 1111 1011 51 46 ff fb 81 70 255 251
search_true_key: 0000 0000 0101 0011 0000 0000 0100 0010 0000 0000 1111 1111 0000 0000 1111 1011 53 42 ff fb 83 66 255 251
search_true_key: 0000 0000 0101 0101 0000 0000 0101 0110 0000 0000 1111 1111 0000 0000 1111 1011 55 56 ff fb 85 86 255 251
search_true_key: 0000 0000 0101 0111 0000 0000 0101 0010 0000 0000 1111 1111 0000 0000 1111 1011 57 52 ff fb 87 82 255 251
search_true_key: 0000 0000 0101 1001 0000 0000 0100 0100 0000 0000 1111 1111 0000 0000 1111 1011 59 44 ff fb 89 68 255 251
search_true_key: 0000 0000 0101 1011 0000 0000 0100 0000 0000 0000 1111 1111 0000 0000 1111 1011 5b 40 ff fb 91 64 255 251
search_true_key: 0000 0000 0101 1101 0000 0000 0101 0100 0000 0000 1111 1111 0000 0000 1111 1011 5d 54 ff fb 93 84 255 251
search_true_key: 0000 0000 0101 1111 0000 0000 0101 0000 0000 0000 1111 1111 0000 0000 1111 1100 5f 50 ff fb 95 80 255 251
search_true_key: 0000 0000 0101 0000 0000 0000 0101 1110 0000 0000 1111 1111 0000 0000 1111 1100 59 5e ff fc 80 94 255 252
search_true_key: 0000 0000 0101 0010 0000 0000 0101 1010 0000 0000 1111 1111 0000 0000 1111 1100 52 5a ff fc 82 90 255 252
search_true_key: 0000 0000 0101 0100 0000 0000 0100 1110 0000 0000 1111 1111 0000 0000 1111 1100 54 4e ff fc 84 78 255 252
search_true_key: 0000 0000 0101 0110 0000 0000 0100 1010 0000 0000 1111 1111 0000 0000 1111 1100 56 4a ff fc 86 74 255 252
search_true_key: 0000 0000 0101 1000 0000 0000 0101 1100 0000 0000 1111 1111 0000 0000 1111 1100 58 5c ff fc 88 92 255 252
search_true_key: 0000 0000 0101 1010 0000 0000 0101 1000 0000 0000 1111 1111 0000 0000 1111 1100 5a 58 ff fc 90 88 255 252
search_true_key: 0000 0000 0101 1100 0000 0000 0100 1100 0000 0000 1111 1111 0000 0000 1111 1100 5c 4c ff fc 92 76 255 252
search_true_key: 0000 0000 0101 1110 0000 0000 0100 1000 0000 0000 1111 1111 0000 0000 1111 1100 5e 48 ff fc 94 72 255 252
search_true_key: 0000 0000 0101 0000 0000 0000 0100 1110 0000 0000 1111 1111 0000 0000 1111 1101 58 4e ff fd 80 70 255 253
search_true_key: 0000 0000 0101 0010 0000 0000 0100 1010 0000 0000 1111 1111 0000 0000 1111 1101 52 4a ff fd 82 74 255 253
search_true_key: 0000 0000 0101 0100 0000 0000 0101 1110 0000 0000 1111 1111 0000 0000 1111 1101 54 5e ff fd 84 94 255 253
search_true_key: 0000 0000 0101 0110 0000 0000 0101 1010 0000 0000 1111 1111 0000 0000 1111 1101 56 5a ff fd 86 90 255 253
search_true_key: 0000 0000 0101 1010 0000 0000 0100 1000 0000 0000 1111 1111 0000 0000 1111 1101 58 4c ff fd 88 76 255 253
search_true_key: 0000 0000 0101 1100 0000 0000 0101 1100 0000 0000 1111 1111 0000 0000 1111 1101 5e 5c ff fd 92 92 255 253
search_true_key: 0000 0000 0101 1110 0000 0000 0101 1000 0000 0000 1111 1111 0000 0000 1111 1101 5f 58 ff fd 94 88 255 253
search_true_key: 0000 0000 0101 0001 0000 0000 0101 1110 0000 0000 1111 1111 0000 0000 1111 1110 51 5e ff fe 81 94 255 254
search_true_key: 0000 0000 0101 0011 0000 0000 0101 1010 0000 0000 1111 1111 0000 0000 1111 1110 53 5a ff fe 83 90 255 254
search_true_key: 0000 0000 0101 0101 0000 0000 0100 1110 0000 0000 1111 1111 0000 0000 1111 1110 55 4e ff fe 85 78 255 254
search_true_key: 0000 0000 0101 0111 0000 0000 0100 1010 0000 0000 1111 1111 0000 0000 1111 1110 57 4a ff fe 87 74 255 254
search_true_key: 0000 0000 0101 1001 0000 0000 0101 1100 0000 0000 1111 1111 0000 0000 1111 1110 59 5c ff fe 89 92 255 254
search_true_key: 0000 0000 0101 1011 0000 0000 0100 1100 0000 0000 1111 1111 0000 0000 1111 1110 5b 58 ff fe 91 88 255 254
search_true_key: 0000 0000 0101 1101 0000 0000 0100 1000 0000 0000 1111 1111 0000 0000 1111 1110 5d 4c ff fe 93 76 255 254
search_true_key: 0000 0000 0101 1111 0000 0000 0100 1000 0000 0000 1111 1111 0000 0000 1111 1110 5f 48 ff fe 95 72 255 254
search_true_key: 0000 0000 0101 0001 0000 0000 0100 1110 0000 0000 1111 1111 0000 0000 1111 1111 51 4e ff ff 81 78 255 255
search_true_key: 0000 0000 0101 0011 0000 0000 0100 1010 0000 0000 1111 1111 0000 0000 1111 1111 53 4a ff ff 83 74 255 255
search_true_key: 0000 0000 0101 0101 0000 0000 0101 1110 0000 0000 1111 1111 0000 0000 1111 1111 55 4e ff ff 85 94 255 255
search_true_key: 0000 0000 0101 0111 0000 0000 0101 1010 0000 0000 1111 1111 0000 0000 1111 1111 57 5a ff ff 87 90 255 255
search_true_key: 0000 0000 0101 1001 0000 0000 0100 1100 0000 0000 1111 1111 0000 0000 1111 1111 59 4c ff ff 89 92 255 255
search_true_key: 0000 0000 0101 1011 0000 0000 0100 1000 0000 0000 1111 1111 0000 0000 1111 1111 5b 48 ff ff 91 72 255 255
search_true_key: 0000 0000 0101 1111 0000 0000 0101 1100 0000 0000 1111 1111 0000 0000 1111 1111 5d 5c ff ff 93 92 255 255
search_true_key: 0000 0000 0101 1111 0000 0000 0101 1000 0000 0000 1111 1111 0000 0000 1111 1111 5f 58 ff ff 95 88 255 255

Keys found: 131072
anfek@anfek: ~/Документы/8_sem/kmzi
```

Рис. 3: Результат.

Затем списком выводится результат выполнения программы.

В нашем случае программа нашла 131072 ключей. Первым делом хочется возмутиться её неэффективностью, но если взглянуть на результат с точки зрения математики – получается следующее:

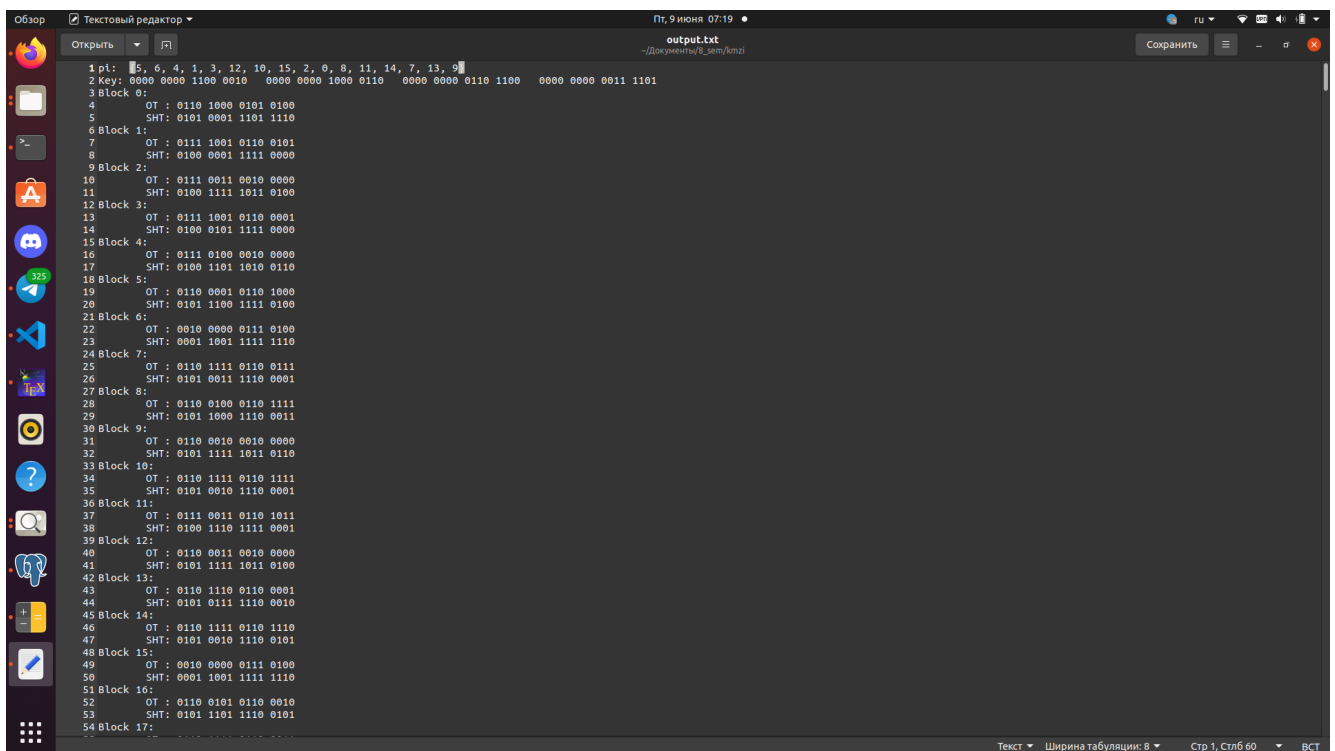
Мы имели ключ длиной 64 бита состоящий из 4-х подключей по 16 бит. Но наши ключи были искусственно ограничены до 8-ми бит. И того у нас получается 32 бита. Таким образом для полного перебора нам бы понадобилось перебрать 2^{32} возможных ключей это 4294967296. На современных компьютерах в целом возможно перебрать 4 миллиарда комбинаций, но затраченное время и ресурсы Вас неприятно удивят. Данная программа находит результат за несколько минут, а для полного перебора Вам понадобится несколько часов или даже больше суток (в зависимости от Ваших вычислительных мощностей и особенностей реализации).

Ещё учтём простоту нашего алгоритма шифрования.

С учётом всего вышесказанного частота коллизий составляет

$$\frac{131072}{4294967296} = \frac{1}{32768} = 0.000030517578125 \approx 3 \cdot 10^{-5}.$$

Теперь проверим наш файл вывода.



```
1 ip: 5, 6, 4, 1, 3, 12, 10, 15, 2, 0, 8, 11, 14, 7, 13, 9
2 Key: 0000 0000 1100 0010 0000 0000 1000 0110 0000 0000 0110 1100 0000 0000 0011 1101
3 Block 0:
4   OT : 0110 1000 0101 0100
5   SHT: 0101 0001 1101 1110
6 Block 1:
7   OT : 0111 1001 0110 0101
8   SHT: 0100 0001 1111 0000
9 Block 2:
10  OT : 0111 0011 0010 0000
11  SHT: 0100 1111 1011 0100
12 Block 3:
13  OT : 0111 1001 0110 0001
14  SHT: 0100 0101 1111 0000
15 Block 4:
16  OT : 0111 0100 0010 0000
17  SHT: 0100 1101 1010 0110
18 Block 5:
19  OT : 0110 0001 0110 1000
20  SHT: 0101 1100 1111 0100
21 Block 6:
22  OT : 0010 0000 0111 0100
23  SHT: 0001 1001 1111 1110
24 Block 7:
25  OT : 0110 1111 0110 0111
26  SHT: 0101 0011 1110 0001
27 Block 8:
28  OT : 0110 0100 0110 1111
29  SHT: 0101 1000 1110 0011
30 Block 9:
31  OT : 0110 0010 0010 0000
32  SHT: 0101 1111 1011 0110
33 Block 10:
34  OT : 0110 1111 0110 1111
35  SHT: 0101 0010 1110 0001
36 Block 11:
37  OT : 0111 0011 0110 1011
38  SHT: 0100 1110 1111 0001
39 Block 12:
40  OT : 0110 0011 0010 0000
41  SHT: 0101 1111 1011 0100
42 Block 13:
43  OT : 0110 1110 0110 0001
44  SHT: 0101 0111 1110 0010
45 Block 14:
46  OT : 0110 1111 0110 1110
47  SHT: 0101 0010 1110 0101
48 Block 15:
49  OT : 0010 0000 0111 0100
50  SHT: 0001 1001 1111 1110
51 Block 16:
52  OT : 0110 0101 0110 0010
53  SHT: 0101 1101 1110 0101
54 Block 17:
```

Рис. 4: Key, ip, blocks.

Он также содержит истинный ключ Key, подстановку ip и пары блоков ОТ-ШТ.

```

Обзор | Текстовый редактор | output.txt | Пт, 9 июня 07:19
Открыть | Сохранить | - | + | x

2985 Block 994:
2986 OT : 0110 1110 0110 1001
2987 SHT: 0101 0110 1110 0010
2988 Block 995:
2989 OT : 0110 1001 0111 0100
2990 SHT: 0101 0001 1111 1100
2991 Block 996:
2992 OT : 0110 0111 0110 1110
2993 SHT: 0101 1010 1110 0101
2994 Block 997:
2995 OT : 0010 1100 0111 0011
2996 SHT: 0001 0101 1110 1011
2997 Block 998:
2998 OT : 0110 0100 0010 0000
2999 SHT: 0101 1101 1010 0110
3000 Block 999:
3001 OT : 0111 0000 0110 0101
3002 SHT: 0100 1001 1111 0010
3003
3004
3005 You key:
3006
3007 search_true_key: 0000 0000 1000 0001 0000 0000 1001 1011 0000 0000 0010 0000 0000 0000 0010 0000 81 9b 20 20 129 155 32 32
3008
3009 search_true_key: 0000 0000 1000 0011 0000 0000 1001 1111 0000 0000 0010 0000 0000 0000 0010 0000 83 9f 20 20 131 159 32 32
3010
3011 search_true_key: 0000 0000 1000 0101 0000 0000 1000 1011 0000 0000 0010 0000 0000 0000 0010 0000 85 8b 20 20 133 139 32 32
3012
3013 search_true_key: 0000 0000 1000 0111 0000 0000 1000 1111 0000 0000 0010 0000 0000 0000 0010 0000 87 8f 20 20 135 143 32 32
3014
3015 search_true_key: 0000 0000 1000 1001 0000 0000 1001 1001 0000 0000 0010 0000 0000 0000 0010 0000 89 99 20 20 137 153 32 32
3016
3017 search_true_key: 0000 0000 1000 1011 0000 0000 1001 1101 0000 0000 0010 0000 0000 0000 0010 0000 8b 9d 20 20 139 157 32 32
3018
3019 search_true_key: 0000 0000 1000 1101 0000 0000 1000 1001 0000 0000 0010 0000 0000 0000 0010 0000 8d 89 20 20 141 137 32 32
3020
3021 search_true_key: 0000 0000 1000 1111 0000 0000 1000 1101 0000 0000 0010 0000 0000 0000 0010 0000 8f 8d 20 20 143 141 32 32
3022
3023 search_true_key: 0000 0000 1000 0001 0000 0000 1000 1011 0000 0000 0010 0000 0000 0000 0010 0001 81 8b 20 21 129 139 32 33
3024
3025 search_true_key: 0000 0000 1000 0011 0000 0000 1000 1111 0000 0000 0010 0000 0000 0000 0010 0001 83 8f 20 21 131 143 32 33
3026
3027 search_true_key: 0000 0000 1000 0101 0000 0000 1001 1011 0000 0000 0010 0000 0000 0000 0010 0001 85 9b 20 21 133 155 32 33
3028
3029 search_true_key: 0000 0000 1000 0111 0000 0000 1001 1111 0000 0000 0010 0000 0000 0000 0010 0001 87 9f 20 21 135 159 32 33
3030
3031 search_true_key: 0000 0000 1000 1001 0000 0000 1000 1001 0000 0000 0010 0000 0000 0000 0010 0001 89 89 20 21 137 137 32 33
3032
3033 search_true_key: 0000 0000 1000 1011 0000 0000 1000 1101 0000 0000 0010 0000 0000 0000 0010 0001 8b 8d 20 21 139 141 32 33
3034
3035 search_true_key: 0000 0000 1000 1101 0000 0000 1001 1001 0000 0000 0010 0000 0000 0000 0010 0001 8d 99 20 21 141 153 32 33
3036
3037 search_true_key: 0000 0000 1000 1111 0000 0000 1001 1101 0000 0000 0010 0000 0000 0000 0010 0001 8f 9d 20 21 143 157 32 33
3038
3039 search_true_key: 0000 0000 1000 0000 0000 0000 1001 1011 0000 0000 0010 0000 0000 0000 0010 0010 80 9b 20 22 128 155 32 34
3040
Текст | Ширина таблицы: 8 | Стр 3002, Стлб 34 | ВСТ

```

Рис. 5: Найденные ключи.

А также ключи, которые нашла программа.
Проверим нашёлся ли искомый истинный ключ.

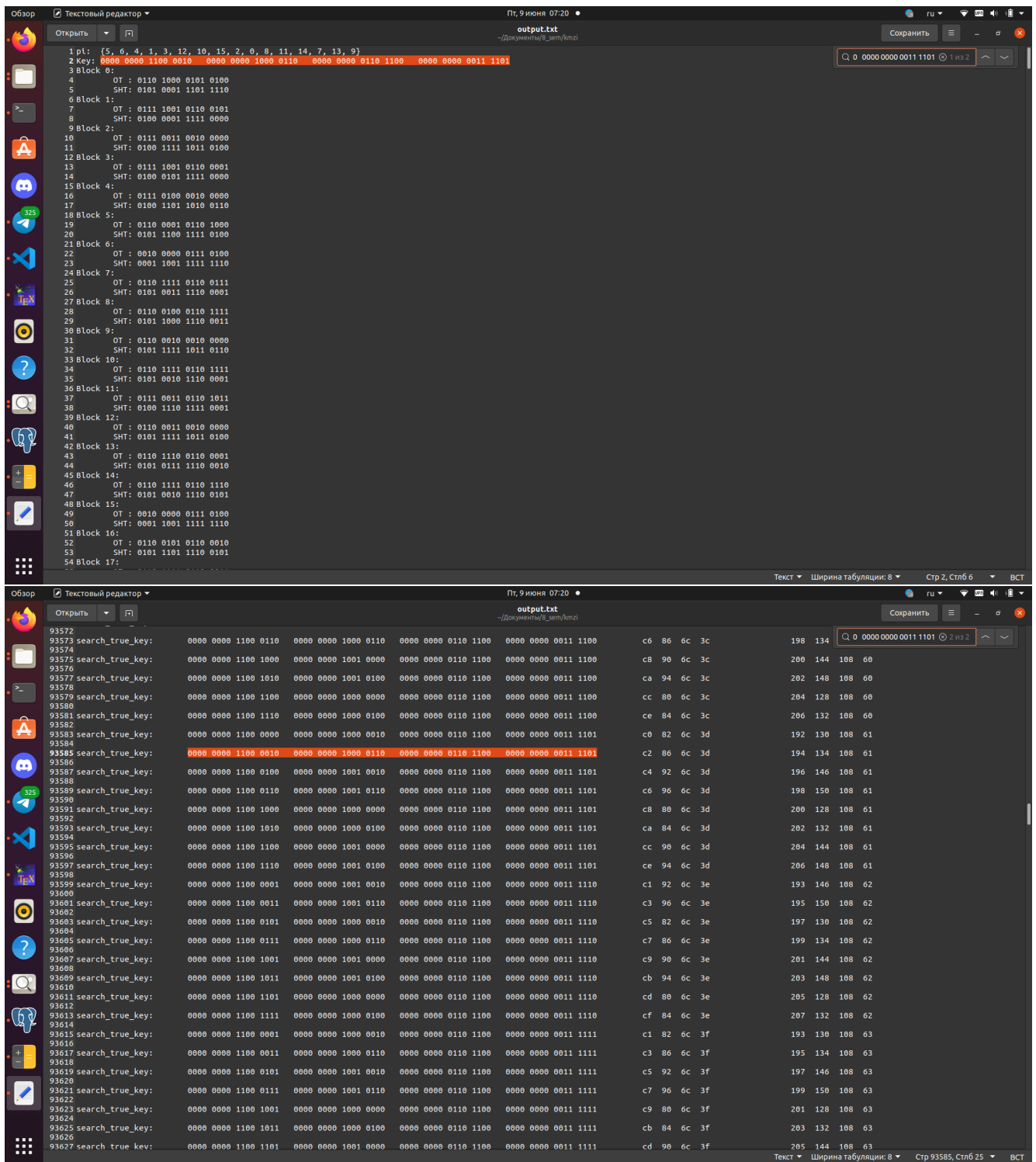


Рис. 6: Проверка существования искомого ключа.

Искомый ключ существует и это говорит о том, что программа работает как нужно. То что остальные ключи являются именно коллизиями к искомому, а не результатами программной ошибки я проверяла как программно «в лоб», так и на бумаге.

3 Заключение

В ходе выполнения данной индивидуальной долгосрочной работы по реализации метода встречи посередине на языке C++, я достигла важных результатов и получил ценный опыт в области криптографических методов защиты информации. Работа над этим проектом позволила мне глубже понять принципы криптографии и применить их на практике.

Основной целью моей работы было создание программы, способной реализовывать метод встречи посередине с использованием 4-х раундового XSP-шифра. Я успешно разработала и реализовала алгоритм шифрования, разделив ключ на раундовые ключи и применив операции X-OR, побитовой перестановки и транспонирования. Благодаря этому, программа обеспечивает безопасное шифрование и дешифрование данных.

Особое внимание было уделено реализации и использованию метода встречи посередине. Я провела анализ пар блоков открытого текста и шифротекста, используя две фазы метода. Перебирая комбинации ключей, генерируя полу-шифротексты и полу-расшифрованные тексты, я создал массивы списков для поиска совпадений. Этот метод позволяет эффективно анализировать безопасность криптографических систем и может быть применен в различных областях, где требуется защита информации.

В результате проведенных исследований и разработки программы я приобрел новые знания и навыки в области криптографии. Эта работа помогла мне глубже понять принципы защиты информации, применение криптографических методов и их влияние на безопасность данных.

Я довольна результатами этой долгосрочной работы, так как она позволила мне не только углубиться в изучение темы криптографии, но и применить полученные знания на практике. Этот проект стал важным шагом в моем образовательном пути и открыл новые перспективы для развития в области информационной безопасности.

Единственным минусом данной программной реализации я считаю огромную ресурсоёмкость по оперативной памяти. При реализации с обычным ключём массивы занимают несколько десятков гигабайт оперативной памяти.

Полную версию кода можно посмотреть в приложении 1.

4 Приложения

4.1 Полная версия кода

```
1 #include <iostream>
2 #include <cstdint>
3 #include <random>
4 #include <array>
5 #include <algorithm>
6 #include <string>
7 #include <fstream>
8
9 #include <unistd.h>
10 #include <deque>
11 #include <chrono>
12
13
14
15 typedef std::array<uint16_t, 4> Type_Key;
16 typedef std::array<uint8_t, 16> Type_Pi;
17
18 std::string output_file = "output.txt";
19
20 struct Block{
21     uint16_t OT, SHT;
22 };
23
24 const int Num_blocks = 1000;
25 typedef std::array<struct Block, Num_blocks> Type_Block_Array;
26
27
28
29 class Crypto{
30 private:
31     Type_Pi pi;
32     Type_Key Key;
33
34
35     void gen_key() {
36         auto current_time = std::chrono::high_resolution_clock::now().time_si
nce_epoch();
37         auto seed = static_cast<unsigned int>(std::chrono::duration_cast<std:
:chrono::milliseconds>(current_time).count());
38
39         std::mt19937 generator(seed);
40
41         std::uniform_int_distribution<uint16_t> distribution(0, 0xFF);
42
43         for (int i = 0; i < 4; ++i) {
44             Key[i] = distribution(generator);
45         }
```

```

46     }
47
48     void shuffleArray(Type_Pi& array) {
49         auto current_time = std::chrono::high_resolution_clock::now().time_si
nce_epoch();
50         auto seed = static_cast<unsigned int>(std::chrono::duration_cast<std:
:chrono::milliseconds>(current_time).count());
51
52         std::mt19937 generator(seed);
53
54         std::shuffle(array.begin(), array.end(), generator);
55     }
56
57     void gen_pi(){
58         for (int i=0; i<pi.size(); i++)
59             pi[i] = i;
60         shuffleArray(pi);
61     }
62
63     uint16_t X_func (uint16_t text, uint16_t round_key){
64         return text ^ round_key;
65     }
66
67     uint16_t S_func_encrypt (uint16_t text){
68         uint16_t result = 0;
69         uint16_t buf = 0x8000;
70         for(int i = 0; i<pi.size(); i++){
71             int sdvig = i - pi[i];
72
73             if(sdvig >= 0)
74                 result |= (buf & text) << sdvig;
75             else
76                 result |= (buf & text) >> (-sdvig);
77             buf >>= 1;
78         }
79         return result;
80     }
81
82     uint16_t S_func_decrypt (uint16_t text){
83         uint16_t result = 0;
84         uint16_t buf; uint8_t index; int sdvig;
85
86         for(int i = 0; i<pi.size(); i++){
87             auto it = std::find(std::begin(pi), std::end(pi), pi[i]);
88             index = it - std::begin(pi);
89
90             sdvig = pi[i] - index;
91             buf = 0x8000 >> pi[i];
92
93             if(sdvig >= 0)
94                 result |= (buf & text) << sdvig;

```

```

95         else
96             result |= (buf & text) >> (-sdvig);
97     }
98     return result;
99 }
100
101 uint16_t P_func (uint16_t text){
102     uint16_t ad = 0xf00f & text;
103     uint16_t b = (0x0f00 & text) >> 4;
104     uint16_t c = (0x00f0 & text) << 4;
105     return ad | b | c;
106 }
107
108 public:
109     void print_bit(uint16_t num, int ch){
110         for(int i = 15; i >= 0; --i) {
111             std::cout << ((num >> i) & 1);
112             if(i%4 == 0)
113                 std::cout << " ";
114         }
115
116         if(ch == 0)
117             std::cout << std::endl;
118     }
119
120     void print_bit_in_file(uint16_t num, int ch, std::ofstream &write_out){
121         for(int i = 15; i >= 0; --i) {
122             write_out << ((num >> i) & 1);
123             if(i%4 == 0)
124                 write_out << " ";
125         }
126
127         if(ch == 0)
128             write_out << "\n";
129     }
130
131     Crypto(){
132         gen_key();
133         gen_pi();
134     }
135     Crypto(Type_Pi pi){
136         this->pi = pi;
137     }
138
139     Type_Pi get_pi(){
140         return pi;
141     }
142     void push_Key(Type_Key Key){
143         this->Key = Key;
144     }
145

```

```

146 void print_pi(){
147     std::cout << "pi:  {";
148     for(int i=0; i<16; i++){
149         std::cout << (int)pi[i];
150         if(i < 15)
151             std::cout << ", ";
152     }
153     std::cout<<"}"<<std::endl;
154 }
155 void print_key(){
156     std::cout << "Key: ";
157     for(int i=0; i<Key.size(); i++){
158         print_bit(Key[i], 1);
159         std::cout << " ";
160     }
161     std::cout << std::endl;
162 }
163
164 void print_key(Type_Key another_key){
165     std::cout << "Key: ";
166     for(int i=0; i<another_key.size(); i++){
167         print_bit(another_key[i], 1);
168         std::cout << " ";
169     }
170     std::cout << std::endl;
171 }
172
173 void print_pi_key_file(){
174     std::ofstream write_out;
175     write_out.open(output_file, std::ios::app);
176
177     write_out << "pi:  {";
178     for(int i=0; i<16; i++){
179         write_out << (int)pi[i];
180         if(i < 15)
181             write_out << ", ";
182     }
183     write_out<<"}\n";
184
185     write_out << "Key: ";
186     for(int i=0; i<Key.size(); i++){
187         print_bit_in_file(Key[i], 1, write_out);
188         write_out << " ";
189     }
190     write_out << "\n";
191
192
193     write_out.close();
194 }
195
196 uint16_t Encryption (uint16_t OT, uint8_t rounde){

```

```

197     uint16_t SHT = OT;
198     for (int i = 0, j = 0; i<Key.size() && j<rounde; i++, j++){
199         SHT = X_func(SHT, Key[i]);
200         SHT = S_func_encrypt(SHT);
201         SHT = P_func(SHT);
202     }
203     return SHT;
204 }
205
206 uint16_t Decryption (uint16_t SHT, uint8_t rounde){
207     uint16_t OT = SHT;
208     for (int i = Key.size()-1, j = 0; i>=0 && j<rounde; i--, j++){
209         OT = P_func(OT);
210         OT = S_func_decrypt(OT);
211         OT = X_func(OT, Key[i]);
212     }
213     return OT;
214 }
215 };
216
217 class Gen_Text_Block{
218 private:
219     std::string File;
220     Type_Block_Array block_array;
221     typedef std::pair<Type_Pi, Type_Block_Array> Type_Return;
222
223
224     int read_text(){
225         std::ifstream inputFile(File, std::ios::binary);
226         if (!inputFile) {
227             std::cout << "Failed to open the file." << std::endl;
228             return 1;
229         }
230
231         const int blockSize = sizeof(block_array[0].OT);
232
233         for (int i = 0; i < Num_blocks; ++i) {
234             inputFile.read(reinterpret_cast<char*>(&block_array[i].OT), block
235 Size);
236             if (!inputFile) {
237                 std::cout << "Failed to read block " << i << " from the file.
238 " << std::endl;
239                 return 1;
240             }
241         }
242
243         inputFile.close();
244         return 0;
245     }
246
247     Type_Pi encryption_blocks(){

```

```

246     Crypto crypto = Crypto();
247     auto pi = crypto.get_pi();
248     crypto.print_key();
249     crypto.print_pi();
250     crypto.print_pi_key_file();
251
252     for (int i=0; i<Num_blocks; i++)
253         block_array[i].SHT = crypto.Encryption(block_array[i].OT, 4);
254     return pi;
255 }
256
257 public:
258     Gen_Text_Block(std::string file_name){
259         File = file_name;
260     }
261     int get_num_blocks(){
262         return Num_blocks;
263     }
264
265     Type_Return gen(){
266         if(read_text())
267             return {{}, block_array};
268         auto pi = encryption_blocks();
269         return {pi, block_array};
270     }
271
272     void print_block_array(){
273         Crypto crypto;
274         std::ofstream write_out;
275         write_out.open(output_file, std::ios::app);
276         for (int i=0; i<Num_blocks; i++) {
277             std::cout << "Block " << i << ":" << std::endl;
278             std::cout << "\tOT : "; crypto.print_bit(block_array[i].OT, 0);
279             std::cout << "\tSHT: "; crypto.print_bit(block_array[i].SHT, 0);
280
281             write_out << "Block " << i << ":\n";
282             write_out << "\tOT : "; crypto.print_bit_in_file(block_array[i].OT, 0, write_out);
283             write_out << "\tSHT: "; crypto.print_bit_in_file(block_array[i].SHT, 0, write_out);
284         }
285         std::cout << std::endl;
286         write_out << "\n";
287         write_out.close();
288     }
289 };
290
291
292 typedef std::array <uint16_t, 2> Type_Two_Key;
293 typedef std::deque <std::vector<Type_Two_Key>> Type_full_list;
294 typedef std::vector <Type_Key> Type_key_list;

```

```

295 class Method_work{
296 private:
297     Type_Pi pi;
298     Type_Block_Array block_array;
299
300
301     Type_key_list list_valid_keys;
302
303     typedef struct {
304         Type_Two_Key key;
305         int lvl;
306     }Thread_Args_First;
307
308
309     void print_key(Type_Two_Key key, std::string name_func){
310         Crypto crypto;
311         std::cout << "\r" << name_func << ":\t";
312         for(int i=0; i<key.size(); i++){
313             crypto.print_bit(key[i], 1);
314             std::cout << " ";
315         }
316         std::cout << "\t";
317         for(int i=0; i<key.size(); i++){
318             std::cout << " " << std::hex << key[i] << std::dec;
319         }
320         std::cout << "\t\t";
321         for(int i=0; i<key.size(); i++){
322             std::cout << " " << key[i];
323         }
324     }
325
326     void print_key(Type_Key key, std::string name_func){
327         Crypto crypto;
328         std::cout << "\r" << name_func << ":\t";
329         for(int i=0; i<key.size(); i++){
330             crypto.print_bit(key[i], 1);
331             std::cout << " ";
332         }
333         std::cout << "\t";
334         for(int i=0; i<key.size(); i++){
335             std::cout << " " << std::hex << key[i] << std::dec;
336         }
337         std::cout << "\t\t";
338         for(int i=0; i<key.size(); i++){
339             std::cout << " " << key[i];
340         }
341     }
342
343     void print_key_in_file(Type_Key key, std::string name_func, std::ofstream
&write_out){
344         Crypto crypto;

```



```

345     write_out << "\r" << name_func << ":\t";
346     for(int i=0; i<key.size(); i++){
347         crypto.print_bit_in_file(key[i], 1, write_out);
348         write_out << " ";
349     }
350     write_out << "\t";
351     for(int i=0; i<key.size(); i++){
352         write_out << " " << std::hex << key[i] << std::dec;
353     }
354     write_out << "\t\t";
355     for(int i=0; i<key.size(); i++){
356         write_out << " " << key[i];
357     }
358 }
359
360 Type_full_list first_phase_1_1 (Type_full_list list_all_keys){
361     Type_Two_Key key;
362     Crypto crypto = Crypto(pi);
363     for(key[0] = 0; key[0]<=0xFF; key[0]++){
364         for(key[1] = 0; key[1]<=0xFF; key[1]++){
365             Type_Key current_Key;
366             std::fill(current_Key.begin(), current_Key.end(), 0);
367             for(int i=0; i<key.size() && i<current_Key.size(); i++)
368                 current_Key[i] = key[i];
369             crypto.push_Key(current_Key);
370             uint16_t e_OT = crypto.Encryption(block_array[0].OT, 2);
371
372             print_key(key, "first_phase_1");
373             list_all_keys[e_OT].emplace_back(key);
374             if(key[1] == 0xff) break;
375         }
376         if(key[0] == 0xff) break;
377     }
378     return list_all_keys;
379 }
380
381 void first_phase_2_1 (Type_full_list list_all_keys){
382     Type_Two_Key key;
383     Crypto crypto = Crypto(pi);
384
385     for(key[0] = 0; key[0]<=0xFF; key[0]++){
386         for(key[1] = 0; key[1]<=0xFF; key[1]++){
387             Type_Key current_Key;
388             std::fill(current_Key.begin(), current_Key.end(), 0);
389             for(int i=key.size()-1, j=current_Key.size()-1; i>=0 && j>=0;
i--, j--)
390                 current_Key[j] = key[i];
391             crypto.push_Key(current_Key);
392             uint16_t d_SHT = crypto.Decryption(block_array[0].SHT, 2);
393
394

```

```

395         if(list_all_keys[d_SHT].size()!=0){
396             for (auto it = list_all_keys[d_SHT].begin(); it != list_a
ll_keys[d_SHT].end(); ++it) {
397                 const auto& element = *it;
398                 for(int i=0; i<element.size() && i<current_Key.size()
; i++)
399                     current_Key[i] = element[i];
400
401                 list_valid_keys.emplace_back(current_Key);
402                 print_key(current_Key, "\t\t KEY first_phase_2");
403             }
404         }
405         if(key[1] == 0xff) break;
406     }
407     if(key[0] == 0xff) break;
408 }
409 }
410
411
412 void first_block(){
413     Type_full_list list_all_keys(0x10000);
414     list_all_keys = first_phase_1_1(list_all_keys);
415     std::cout << "\n\nfirst_phase_1 list all: " << list_all_keys.size() <
< "\t" << std::hex << list_all_keys.size() << std::dec << std::endl;
416
417
418     first_phase_2_1(list_all_keys);
419     std::cout << "\n\nfirst_phase_2 list val: " << list_valid_keys.size()
<< "\t" << std::hex << list_valid_keys.size() << std::dec << std::endl;
420 }
421
422 void other_rekurs(const int num){
423     Crypto crypto = Crypto(pi);
424     auto element = list_valid_keys.begin();
425     while(element != list_valid_keys.end()){
426         crypto.push_Key(*element);
427         uint16_t e_OT = crypto.Encryption(block_array[num].OT, 2);
428         uint16_t d_SHT = crypto.Decryption(block_array[num].SHT, 2);
429         if(e_OT == d_SHT)
430             ++element;
431         else
432             element = list_valid_keys.erase(element);
433     }
434     std::cout << "other_rekurs: num = " << num << "\tlist_valid_keys.size(
) = " << list_valid_keys.size() << "\t" << std::hex << list_valid_keys.size() << std::d
ec << std::endl;
435 }
436
437 void other_blocks(){
438     for(int i=1; i<Num_blocks; i++){
439         other_rekurs(i);

```

```

440
441     }
442 }
443
444
445 public:
446     Method_work(Type_Pi pi, Type_Block_Array block_array){
447         this->pi = pi;
448         this->block_array = block_array;
449     }
450
451     void search_true_key(){
452         first_block();
453         std::cout << std::endl;
454
455         other_blocks();
456         std::cout << std::endl;
457
458
459         std::ofstream write_out;
460         write_out.open(output_file, std::ios::app);
461
462         std::cout << "\nYou key: " << std::endl;
463         write_out << "\nYou key: \n";
464         for(auto& it : list_valid_keys){
465             print_key(it, "search_true_key");
466             print_key_in_file(it, "search_true_key", write_out);
467             std::cout << std::endl;
468             write_out << "\n";
469         }
470         std::cout << "\nKeys found: "<< list_valid_keys.size() << std::endl;
471         write_out.close();
472     }
473 };
474
475
476 int main(){
477     std::fstream clear_file(output_file, std::ios::out);
478     clear_file.close();
479
480     Gen_Text_Block text("text.txt");
481     auto return_pi_and_array = text.gen();
482     text.print_block_array();
483
484
485     Method_work work(return_pi_and_array.first, return_pi_and_array.second);
486     work.search_true_key();
487
488     return 0;
489 }

```