# Содержание

## Введение

В настоящее время проблема информационной безопасности и защиты обрабатываемой информации становится все более актуальной. Каждый пользователь компьютера стремится обезопасить свои личные данные [1].

Проблемой обеспечения конфиденциальности, целостности и доступности данных занимается криптография. Для защиты информации от посторонних лиц обычно применяется шифрование, которое преобразовывает данные в нечитаемый набор символов по определённому алгоритму с помощью ключа. Если злоумышленник перехватит секретный (закрытый) ключ, то он сможет получить доступ к зашифрованной информации. Salsa20 — система поточного шифрования, разработанная Даниэлем Бернштейном. Алгоритм был представлен на конкурсе «eSTREAM», целью которого было создание европейских стандартов для шифрования данных, передаваемых почтовыми системами. Алгоритм стал победителем конкурса в первом профиле.

Целью работы является программная реализация и криптоанализ алгоритма поточного шифрования Salsa20.

Цель работы определила задачи работы:

– изучить теоретические основы криптографических систем шифрования и их классификацию;

– изучить алгоритм Salsa20;

– криптоанализ на основе атаки на алгоритм

Объектом курсовой работы являются симметрические криптосистемы.

– Предметом курсовой работы является криптоанализ на основе атаки на алгоритм симметричной системы Salsa20.

Информационной базой работы является: монографии, научные статьи по теме исследования, официальная документация к языку программирования python.

Методологическая основа курсовой работы включает следующие научные методы: анализ, дедукция, сравнение, описание и программная реализация.

# 1 Теоретические основы криптографических систем шифрования

В зависимости от количества используемых ключей алгоритмы шифрования делятся на 2 типа:

1. система с закрытым ключом (симметричная). Для шифрования и расшифрования текста используется один закрытый ключ. Его необходимо держать в секрете и передавать только через защищенные каналы связи, так как при перехвате или потере ключа злоумышленник сможет получить доступ к секретной информации;

2. система с открытым ключом (асимметричная). В ней используется пара ключей, которая представляет собой связку из открытого и закрытого ключа. Первый из них необходим для шифрования исходного сообщения. Он передаётся по открытым каналам связи и может являться общедоступным. Закрытый ключ необходим для расшифровки шифротекста, его необходимо держать в секрете. К этой системе также можно отнести алгоритм электронной подписи (ЭП). Для ее генерации используется закрытый ключ, а для проверки ЭП – открытый.

Ассиметричное шифрование на сегодняшний день является более актуальным и криптостойким, однако, в алгоритмах данной системы применяются сложные математические операции в конечных полях над большими числами, например, возведение в степень и умножение двух простых больших чисел по модулю, поиск первообразных корней или квадратичных вычетов. Эти операции требуют больших вычислительных ресурсов компьютера, поэтому ассиметричные криптосистемы уступают по скорости шифрования симметричным. Из этого следует, что для шифрования данных обычно применяются системы с закрытым ключом, а для электронной подписи, протокола обмена ключами, установки защищённого соединения, шифрования закрытого ключа симметричного шифра, а также подписания и проверки сертификатов используются криптосистемы с открытым ключом.

## 1.1 Симметричные криптосистемы

До 1970 года симметричные шифры являлись единственным видом шифрования [2]. Они применялись на всех четырёх этапах развития криптографии. Самые простые шифры перестановки появились ещё до нашей эры, например: шифр Цезаря, Полибианский Квадрат, шифр Атбаш. Их суть заключается в замене символов открытого текста в соответствии с некоторым правилом, например, замена на буквы того же алфавита с некоторым сдвигом. Криптоанализ этих шифров можно успешно провести ручным способом за небольшой промежуток времени.

Современные симметричные шифры делятся на 2 типа:

1. Блочные алгоритмы шифрования представляют собой шифры оперирующие группами бит фиксированной длины — блоками, характерный размер которых меняется в пределах 64–256 бит [3]. Большинство блочных шифров основаны на сети Фейстеля;

2. Потоковые или поточные алгоритмы шифрования представляют собой шифр, в котором каждый символ открытого текста преобразуется в символ шифрованного текста в зависимости не только от используемого ключа, но и от его расположения в потоке открытого текста. Поточный шифр реализует другой подход к симметричному шифрованию, нежели блочные шифры.

## 1.2 Асимметричные криптосистемы

Асимметричное шифрование с открытым ключом базируется на следующих принципах:

1. можно сгенерировать пару очень больших чисел (открытый ключ и закрытый ключ) так, чтобы, зная открытый ключ, нельзя было вычислить закрытый ключ за разумный срок.

2. имеются надёжные методы шифрования, позволяющие зашифровать

сообщение открытым ключом так, чтобы расшифровать его можно было только закрытым ключом. Механизм шифрования является общеизвестным.

3. владелец двух ключей никому не сообщает закрытый ключ, но передает открытый ключ контрагентам или делает его общеизвестным.

Примеры ассиметричных систем: RSA, DSA, алгоритм обмена ключами Диффи-Хелмана, система Эль-Гамаля, ГОСТ Р 34.10-2012.

Одной из первых ассиметричных криптосистем является RSA. Алгоритм был придуман в 1977 году учеными Ривестом, Шамиром и Адлеманом [5]. Несмотря на то, что RSA был изобретен 45 лет назад, он до сих пор остается актуальным и применяется в таких криптографических протоколах, как TLS, S/MIME, PGP. Криптосистема основывается на вычислительной сложности факторизации больших целых чисел, что относится к классу NP задач [5].

Вывод по разделу

Таким образом, в зависимости от количества ключей различают два вида криптографических систем: симметричные и асимметричные.

На основании рассмотренных алгоритмов, в рамках курсовой работы будет реализован алгоритм Salsa20, чья криптостойкость основывается на дифференциальном криптоанализе, многократному сдвигу, сложению по модулю $2^{32}$ , а также проведен его криптоанализ.

# 2 Анализ подхода к криптоанализу алгоритма Salsa20

## 2.1 Криптоанализ системы Salsa20

В 2005 году Paul Crowley объявил об атаке на Salsa20/5 с расчетной сложностью по времени $2^{165}$. Эта и последующие атаки основаны на усеченном дифференциальном криптоанализе. За этот криптоанализ он получил награду в 1000$ от автора Salsa20.

В 2006, Fischer, Meier, Berbain, Biasse, и Robshaw сообщили об атаке на Salsa/6 со сложностью $2^{117}$, и об атаке со связанными ключами на Salsa20/7 со сложностью $2^{217}$.

В 2008 году Aumasson, Fischer, Khazaei, Meier и Rechberger сообщили об атаке на Salsa20/7 со сложностью $2^{153}$ и о первой атаке на Salsa20/8 со сложностью $2^{251}$.

Пока что не было публичных сообщений об атаках на Salsa20/12 и полную Salsa20/20.

Дифференциальном криптоанализ - метод криптоанализа симметричных блочных шифров (и других криптографических примитивов, в частности, хеш-функций и поточных шифров). Дифференциальный криптоанализ основан на изучении преобразования разностей между шифруемыми значениями на различных раундах шифрования. В качестве разности, как правило, применяется операция побитового суммирования по модулю 2, хотя существуют атаки и с вычислением разности по модулю $2^{32}$. Является статистической атакой. Применим для взлома DES, FEAL и некоторых других шифров, как правило, разработанных ранее начала 90-х. Количество раундов современных шифров (AES, Camellia и др.) рассчитывалось с учётом обеспечения стойкости, в том числе и к дифференциальному криптоанализу.

В криптографии усеченный дифференциальный криптоанализ является обобщением дифференциального криптоанализа, атаки на блочные шифры. Ларс Кнудсен разработал методику в 1994 году. В то время как обычный

дифференциальный криптоанализ анализирует полное различие между двумя текстами, усеченный вариант учитывает различия, которые определены только частично. То есть атака предсказывает только некоторые биты вместо полного блока. Этот метод был применен к SAFER, IDEA, Skipjack, E2, Twofish, Camellia, CRYPTON и даже к потоковому шифру Salsa20.

### 2.2 Алгоритмическое конструирование атаки на связанных ключах

Атака на связанных ключах — вид криптографической атаки, в которой криптоаналитик выбирает связь между парой ключей, но сами ключи остаются ему неизвестны. Данные шифруются обоими ключами. В варианте с известным открытым текстом криптоаналитику известны открытый текст и шифротекст данных, шифрованных двумя ключами. Цель злоумышленника состоит в том, чтобы найти фактические секретные ключи. Предполагается, что атакующий знает или выбирает некоторое математическое отношение, связывающее между собой ключи. Соотношение имеет вид:

$$K_B = F(K_A)$$

где F — функция, выбранная атакующим, $K_A$ и $K_B$ — связанные ключи. К каждому шифрованию соотношение между ключами подбирается индивидуально. Существует много способов найти это соотношение правильно. По сравнению с другими атаками, в которых атакующий может манипулировать только открытым текстом и/или зашифрованным текстом, выбор соотношения между секретными ключами даёт дополнительную степень свободы для злоумышленника. Недостатком этой свободы является то, что такие нападения могут быть более трудными на практике. Тем не менее, проектировщики обычно пытаются создать «идеальные» примитивы, которые могут быть автоматически использованы без дальнейшего анализа в максимально широком наборе протоколов или режимов работы. Таким образом, сопротивление таким атакам является важной целью проектирования блочных шифров, и фактически это была одна из заявленных целей проектирования алгоритма Rijndael.

Классическая атака на связанных ключах, данный тип атаки впервые был предложен израильским учёным Эли Бихамом в 1992 году в статье New Types of Cryptanalytic Attacks Using Related Keys. Атака на связанных ключах, описанная им, напоминает сдвиговую атаку. Сдвиговая атака— криптографическая атака, являющаяся, в общем случае, атакой на основе подобранного открытого текста, позволяющая проводить криптоанализ блоковых многораундовых шифров независимо от количества используемых раундов. Предложена Алексом Бирюковым и Дэвидом Вагнером в 1999 году. Сдвиговая атака использует два открытых текста $P$ и $P'$, удовлетворяющих следующему соотношению:

$$P' = F(P, k_1),$$

где $F()$— функция раунда, а $k_1$ — подключ 1-го раунда. В атаке на связанных ключах применяется похожее соотношение между ключами. Допустим, что искомый ключ шифрования K после модификации его процедурой расширения ключа $E_x(K)$ даёт последовательность подключей:

$$Ex(K) = \{k_1, k_2, \ldots, k_{r-1}, k_r\}$$

где r — количество раундов алгоритма шифрования. Предположим, что существует ключ K', расширение которого даёт следующую последовательность:

$$Ex(K') = \{k_2, k_3, \ldots, k_r, k_1\}$$

то есть последовательность подключей, создаваемая на основе ключа $K'$, циклически сдвинута относительно последовательности искомого ключа $K$ на 1 раунд.

Суть атаки:

1. Предположим, что криптоаналитику известно $2^{\frac{n}{2}}$ пар $(P, C)$ открытых текстов и соответствующих им шифротекстов (зашифрованных при помощи ключа $K$), где n — размер блока шифруемых данных, представленного в битах.

2. Кроме того, криптоаналитику известно $2^{\frac{n}{2}}$ пар текстов $(P', C')$ зашифрованных с использованием ключа $K'$, связанного с $K$ приведённым выше соотношением.

3. Для каждого соотношения текстов $(P, C)$ и $(P', C')$ криптоаналитику необходимо найти решения системы уравнений

$$\begin{cases} F(P, k*) = P' \\ F(C, k*) = C' \end{cases}$$

Какой из $2^{\frac{n}{2}}$ текстов $P$ соответствует каждому тексту из $P$, криптоаналитик не знает заранее. Но, вероятность того, что два случайно выбранных текста будут удовлетворять требуемому соотношению, равна .Тогда требуемая пара должна быть найдена после анализа не более чем $2^{\frac{n}{2}+1}$ текстов, в соответствии с парадоксом дней рождения. Условие $2^{\frac{n}{2}+1}$ текстов не является строгим, оно является оценочным и будет выполняться лишь в среднем.

Ключ $k*$, найденный из приведённой выше системы, и есть искомый подключ $k_1$. В зависимости от операции формирования ключа, знание $k_1$ может дать криптоаналитику много возможностей для осуществления несанкционированного доступа к информации. Например, формирование ключа алгоритма LOKI89 построено таким образом, что $k_1$ представляет собой просто левую 32-битную часть ключа $K$. Поскольку в данном алгоритме используется 64-битный ключ, то после вычисления $k_1$ для нахождения $K$ достаточно всего лишь перебора $2^{32}$ вариантов.

Функция раунда $F()$ алгоритма, на который производится атака, должна быть достаточно слабой для вычисления $k*$, что применительно к большинству современных алгоритмов шифрования. Кроме того, трудоёмкость атаки может быть значительно снижена по отношению к описанному выше случаю, все зависит от выбранного алгоритма шифрования открытого текста.

Вывод по разделу

Таким образом, в данном разделе была описан подход к криптоанализу симметричной системы salsa20. Ее криптографическая стойкость основывается на сложности дифференциального криптоанализа. Алгоритм подвержен атаке со связанными ключами и атаке, основанной на усеченном дифференциальном криптоанализе, многократному сдвигу регистра и сложению по модулю 2.

# 3 Программная реализация системы salsa20

## 3.1 Алгоритм работы программного средства

Алгоритм работы протокола Salsa20 представлен в виде блок-схемы на рисунке.

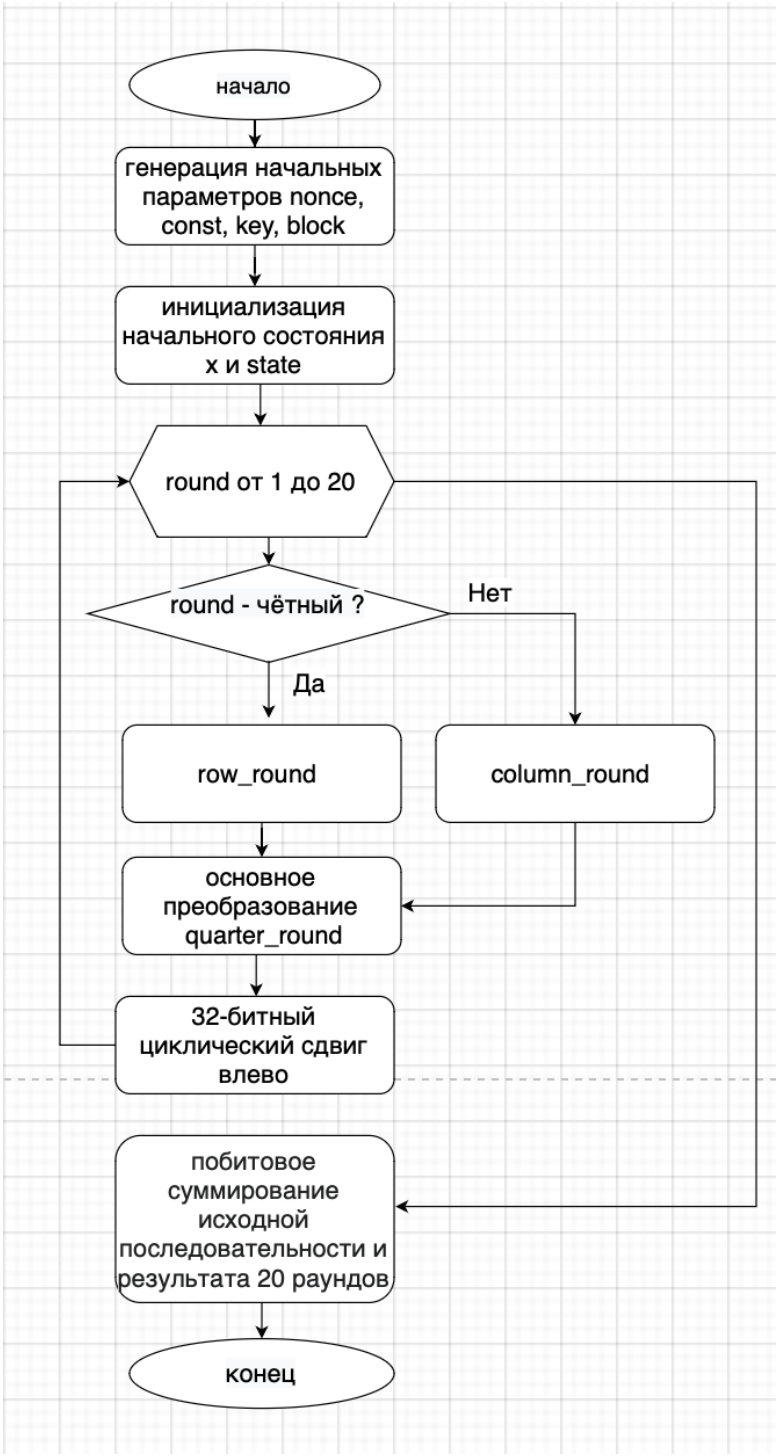


Рис. 1 - Алгоритм работы протокола Salsa20

### 3.2 Описание работы программы

Основным инструментом разработки программы был выбран язык программирования Python.

Достоинствами языка Python являются:

– он поддерживает множество парадигм программирования, в том числе объектно-ориентированное, функциональное, императивное, структурное, процедурное, логическое, контрактное, аспектно-ориентированное, метапрограммирование и т.д.

– Программирование на Python позволяет разбивать программы на составные части – модули, которые можно объединять в пакеты.

– кроссплатформенность. Python – это интерпретируемый язык, его интерпретаторы существуют для многих платформ. Поэтому с запуском его на любой ОС не должно возникнуть проблем;

– с Python доступно огромное количество сервисов, сред разработки, и фреймворков. Легко можно найти подходящий продукт для работы;

– возможность подключить библиотеки, написанные на языке C и C++. Это позволяет повысить эффективность и улучшить быстродействие [11];

– легкий и понятный синтаксис, который позволяет быстро понять, а также дополнить код программы;

– предоставляет возможность для научных вычислений, а также для визуализации данных и построения графиков. Математические операции удобно выполняются в Python из-за его внимания к минимализму в сочетании с полезностью.

Программа написана в объектно-ориентированном стиле и содержит следующие классы:

- Salsa20 – реализует шифрование и расшифрование переданного сообщения на основе алгоритма Salsa20. Содержит следующие методы:

1. __init__– инициализирует объект, принимает на вход ключ и

случайную последовательность nonce, проверяя параметры на корректность. Генерирует необходимые константы для работы алгоритма;

2. initial_state – производит заполнение начальной матрицы X;

3. little_endian – функция преобразования 32-битного слова по специальной формуле;

4. rounds – выполнение 20 раундов алгоритма;

5. column_round – преобразует текущее состояние в матрицу 4x4, далее каждый столбец передаётся в функцию quarter_round;

6. row_round – преобразует текущее состояние в матрицу 4x4, далее каждая строка передаётся в функцию quarter_round;

7. quarter_round – основное преобразование алгоритма. Для каждого слова суммируются два предыдущих, далее полученная сумма циклично сдвигается на определённое количество бит и побитово складывается с выбранным словом;

8. rotate – 32-битовый циклический левый сдвиг слова;

9. encrypt – функция производит шифрование переданного сообщения;

10. decrypt – функция производит расшифрование переданного сообщения;

11. get_random_string – генерирует случайную строку заданной длины;

### 3.3 Тестирование программного средства

Для демонстрации работы программы консольный интерфейс. Перед запуском Криптосистемы Salsa20 необходимо ввести сообщение длиной до 32 символов, которое будет передаваться в зашифрованном виде от абонента A к абоненту B. Далее требуется нажать кнопку «enter». Программное средство сгенерирует необходимые начальные значения: ключ, случайную последовательность для шифрования. Затем происходит генерация константного значения и блока значений необходимых для корректной работы функций. Результат работы системы Salsa20 представлен на рисунке 6.

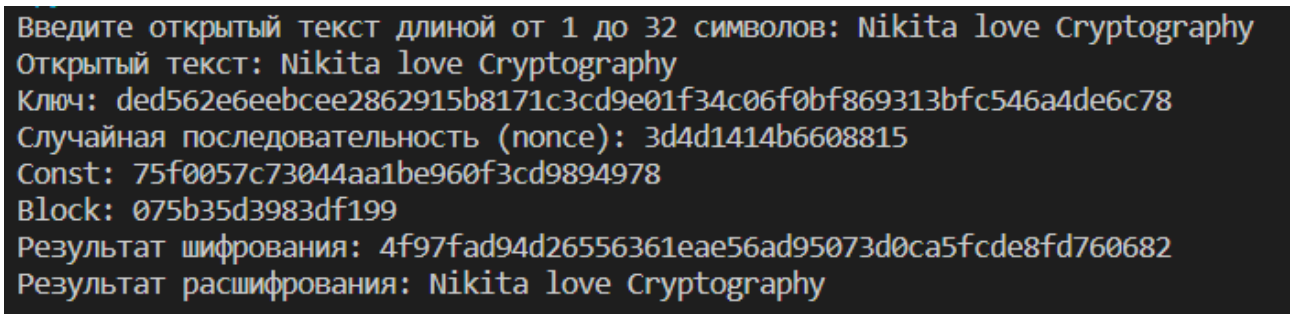


Рисунок 6 – Результат работы криптосистемы Salsa20

Вывод по разделу

Таким образом, программное средство имеет понятный для пользователя интерфейс, руководство использования и описание исходного кода, а также соответствует заявленным требованиям задания на курсовую работу.

# Заключение

Результатом проделанной работы является программная реализация и криптоанализ алгоритма Salsa20.

По ходу выполнения работы были изучены теоретические основы криптографических систем шифрования и их классификации с подробным описанием симметричных и ассиметричных криптосистем, изучен алгоритм Salsa20 и реализована атака на связанных ключах.

В ходе проделанной работы были проанализированы и пошагово описан алгоритм построения криптосистемы Salsa20. Разработано программное средство криптосистемы Salsa20 с полным функционалом. В программе успешно реализована: генерация ключа, генерация случайной последовательности, константы и блока для вычисления функций, а также система шифрования и дешифрования сообщения. В результате данной работы можно отметить, что данная программная реализация является актуальной и в настоящее время так как данная криптосистема, так и не была взломана до сих пор.

Таким образом, поставленные в курсовой работе задачи были выполнены, а цели достигнуты.

## Список используемых источников


1. Симметричное шифрование / Лаборатория Касперского: [сайт]. — URL: https://encyclopedia.kaspersky.ru/glossary/symmetric-encryption. (дата обращения: 5.11.2022).
2. Саймон С. Книга шифров. Тайная история шифров и их расшифровки / С. Саймон — Москва : Астрель, 2007. — 448 с. (дата обращения: 15.05.2022).
3. Алгоритмы шифрования. Симметричный алгоритм шифрования основные понятия [сайт]. — URL: http://more-it.ru/algoritmy-shifrovaniya-simmetrichnyj-algoritm-shifrovaniya-osnovnye-ponyatiya (дата обращения: 8.10.2022).
4. Омассон Жан-Филипп. О криптографии всерьез - М.: ДМК Пресс, 2022. – 328 с.
5. Атака человек «человек посередине» [сайт]. — URL: https://en.wikipedia.org/wiki/Man-in-the-middle_attack (дата обращения: 16.10.2022).
6. Шнайер, Б. Прикладная криптография. Протоколы, алгоритмы и исходный код на С / Б. Шнайер — Москва : Диалектика, 2016. — 1024 с.
7. Функция Эйлера [сайт]. — URL: http://e-maxx.ru/algo/export_euler_function (дата обращения: 20.05.2022).
8. Salsa20 [сайт]. — URL: Salsa20 - Wikipedia (дата обращения: 25.11.2022).
9. Дифференциальный криптоанализ [сайт]. — URL: Дифференциальный криптоанализ — Википедия (wikipedia.org) (дата обращения: 25.11.2022).
10. Атака на связанных ключах [сайт]. — URL: Атака на связанных ключах — Википедия (wikipedia.org) (дата обращения: 25.11.2022).

# Приложение А

Листинг А.1 – Исходный код файла diffi.py

```
import random
class Salsa20:

    def __init__(self, key, nonce):
        if len(key) != 64:
            raise ValueError('Invalid key length, must be 64 byte!')
        self.key = key

        if len(nonce) != 16:
            raise ValueError("Invalid nonce, must be 16 byte!")
        self.nonce = nonce

        self.round = 20
        # self.const = "657870616e642033322d62797465206b"
        self.const = get_random_string(32)  # random
        # self.block = "1ff0203f0f535da1"
        self.block = get_random_string(16)  # random
        self.x = []
        self.state = []
        self.result = []  # 128 чисел в 16сс по 4 бита
        self.initial_state()
        self.rounds()

    def initial_state(self):
        self.x.append(self.little_endian(int(self.const[0:8], 16)))
        # x[1]
        self.x.append(self.little_endian(int(self.key[0:8], 16)))
        # x[2]
        self.x.append(self.little_endian(int(self.key[8:16], 16)))
        # x[3]
        self.x.append(self.little_endian(int(self.key[16:24], 16)))
        # x[4]
        self.x.append(self.little_endian(int(self.key[24:32], 16)))
        # x[5]
        self.x.append(self.little_endian(int(self.const[8:16], 16)))
        # x[6]
        self.x.append(self.little_endian(int(self.nonce[0:8], 16)))
```

```
        # x[7]
        self.x.append(self.little_endian(int(self.nonce[8:16], 16)))
        # x[8]
        self.x.append(self.little_endian(int(self.block[0:8], 16)))
        # x[9]
        self.x.append(self.little_endian(int(self.block[8:16], 16)))
        # x[10]
        self.x.append(self.little_endian(int(self.const[16:24], 16)))
        # x[11]
        self.x.append(self.little_endian(int(self.key[32:40], 16)))
        # x[12]
        self.x.append(self.little_endian(int(self.key[40:48], 16)))
        # x[13]
        self.x.append(self.little_endian(int(self.key[48:56], 16)))
        # x[14]
        self.x.append(self.little_endian(int(self.key[56:64], 16)))
        # x[15]
        self.x.append(self.little_endian(int(self.const[24:32], 16)))

        self.state = self.x[:]

    def little_endian(self, a):
        b = list(range(4))
        # print(a)
        # запись с младшего байта
        b[0] = a >> 24 & 0xff  # & 0xff для того, чтобы не выйти за границу 32 бита
        b[1] = (a >> 16) & 0xff
        b[2] = (a >> 8) & 0xff
        b[3] = a & 0xff

        res = b[0] + 2 ** 8 * b[1] + 2 ** 16 * b[2] + 2 ** 24 * b[3]  # 32-бита
        # print(f'bin_len = {len(bin(res)[2:])}, res = {res}, bin = {bin(res)[2:]}')
        return res

    def rounds(self):
        for i in range(0, self.round, 2):
            self.column_round()
            self.row_round()

        for i in range(16):
            little = self.little_endian(self.x[i] + self.state[i])
```

```
            for j in range(7):
                self.result.append(little >> (32 - 4 * j) & 0xf)

    def row_round(self):
        self.quarter_round(0, 1, 2, 3)  # row 1
        self.quarter_round(5, 6, 7, 4)  # row 2
        self.quarter_round(10, 11, 8, 9)  # row 3
        self.quarter_round(15, 12, 13, 14)  # row 4

    def column_round(self):
        self.quarter_round(0, 4, 8, 12)  # column 1
        self.quarter_round(5, 9, 13, 1)  # column 2
        self.quarter_round(10, 14, 2, 6)  # column 3
        self.quarter_round(15, 3, 7, 11)  # column 4

    def quarter_round(self, x0, x1, x2, x3):
        self.x[x1] ^= self.rotate((self.x[x0] + self.x[x3]), 7)
        self.x[x2] ^= self.rotate((self.x[x1] + self.x[x0]), 9)
        self.x[x3] ^= self.rotate((self.x[x2] + self.x[x1]), 13)
        self.x[x0] ^= self.rotate((self.x[x3] + self.x[x2]), 18)

    @staticmethod
    def rotate(a, b):
        return ((a << b) & 0xffffffff) | (a >> (32 - b))

    def encrypt(self, plaintext):
        plaintext = plaintext.encode("utf-8").hex()  # в 16 сс
        output = ""
        total_byte = len(plaintext)
        for i in range(total_byte):
            output += format(self.result[i] ^ int(plaintext[i: i + 1], 16), "x")

        return output

    def decrypt(self, ciphertext):
        total_byte = len(ciphertext)
        output = ""

        for i in range(total_byte):
            output += format(self.result[i] ^ int(ciphertext[i: i + 1], 16), "x")
```

```
        return bytes.fromhex(output).decode('utf-8')

def get_random_string(length):
    random_values = list(range(10)) + ['a', 'b', 'c', 'd', 'e', 'f']
    res = random.choices(random_values, k=length)
    return ''.join(list(map(str, res)))

if __name__ == '__main__':
    # string = 'Nikita EGOROV'
    # string = ''
    # key = "4c3752b70375de25bfbbea8831edb330ee37cc244fc9eb4f03519c2fcb1af4f3"
    # nonce = "afc7a6305610b3cf"

    string = input('Введите открытый текст длиной от 1 до 32 символов: ')
    if not 0 < len(string) <= 32:
        raise ValueError('Открытый текст должен иметь длину от 1 до 32 символов')
    key = get_random_string(64)
    nonce = get_random_string(16)
    salsa = Salsa20(key, nonce)
    enc = salsa.encrypt(string)
    dec = salsa.decrypt(enc)
    assert string == dec
    print(f'Открытый текст: {string}\nКлюч: {key}\nСлучайная последовательность
(nonce): {nonce}\nConst: {salsa.const}'
          f'\nBlock: {salsa.block}\nРезультат шифрования: {enc}\nРезультат
расшифрования: {dec}\n')
```