Министерство образования и науки Российской Федерации

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н.Г.ЧЕРНЫШЕВСКОГО»

Кафедра физики открытых систем
наименование кафедры

Реализация алгоритмов блочного шифрования

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

Студента 4 курса 431 группы

направления 09.03.02 «Информационные системы и технологии»
код и наименование направления

факультета нелинейных процессов
наименование факультета

Федорова Антона Николаевича
фамилия, имя, отчество

Научный руководитель
профессор, д.ф.-м.н _____ А.Н. Павлов
должность, уч. степень, уч. звание дата, подпись инициалы, фамилия

Зав. кафедрой:
д.ф.-м.н., профессор _____ А.А. Короновский
должность, уч. степень, уч. звание дата, подпись инициалы, фамилия

Саратов 2019 г.

Содержание

Введение

В век компьютерных технологий шифрование данных является весьма актуальной проблемой. В сети интернет передаются достаточно объемные данные - денежные транзакции, приватные сообщения и еще какая либо информация, которую желательно скрыть от третьих лиц. Так как информация нуждается в защите, были разработаны и продолжают разрабатываться алгоритмы шифрования. В данной работе будет рассматриваться блочное шифрование, в частности алгоритм RC5.

Так же будет произведено сравнение времени затраченного программой на реализацию алгоритмов и оптимизация программы для уменьшения требуемого времени.

1. Алгоритм RC5

Алгоритм RC5 представляет из себя блок-шифр с симметричным ключом, являющейся достаточно простым. Разработал его Рональд Ривест в 1994 году. Его «последователь» RC6 Advanced Encryption Standard (AES) был основан на самом RC5.

В отличие от других схем, RC5 имеет возможность менять размер блока (32, 64 или 128 бит), размер ключа (от 0 до 2040 бит), а так же количество раундов (от 0 до 255). Изначально предлагаемым выбором параметров был размер блока 64 бит, 128-битный ключ и 12 раундов.

Ключевой особенностью RC5 является использование зависящих от данных ротаций. RC5 также состоит из нескольких сложений по модулю и операции исключающего ИЛИ (XOR). Общая структура алгоритма - это сеть, подобная сети Фейстеля. Шифрование и дешифрование могут быть реализованы всего в нескольких строках кода.

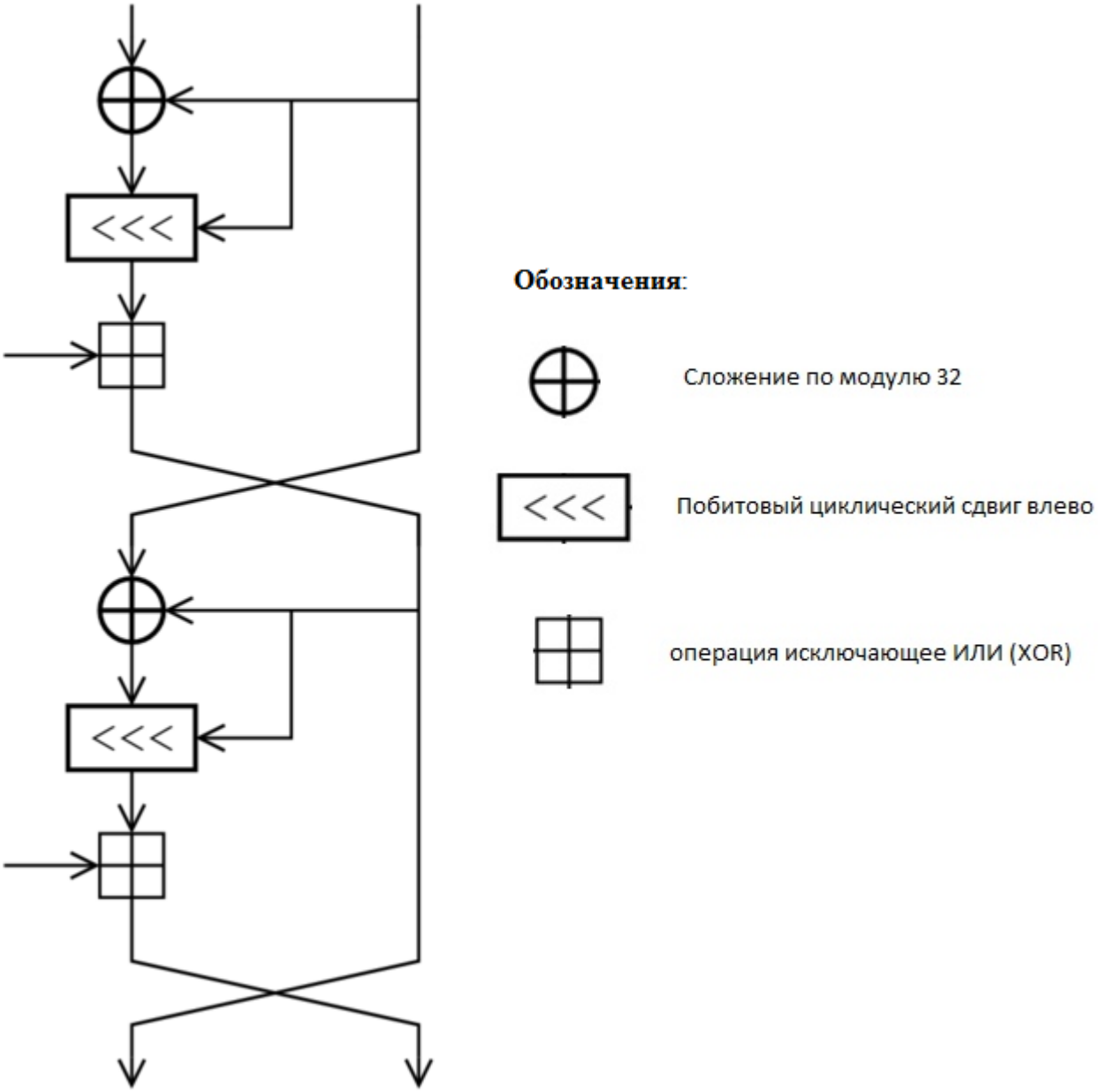

Рисунок 1 - Блок-схема одного раунда в RC5

Алгоритм состоит из трех функций: расширение ключа, шифровка, дешифровка. Можно представить их в виде небольшого псевдокода, взятого из оригинальной статьи Ривеста (комментарии переведены) [2]:

```
# Функция расширения ключа:
# Разбиваем ключ K на слова
# u = w / 8
c = ceiling( max(b, 1) / u )
# L изначально список длиной c заполненный нулями длины w бит
```

```
for i = b-1 down to 0 do:
   L[i/u] = (L[i/u] << 8) + K[i]
# Инициализируем независимый от ключа псевдослучайный массив S
# S изначально лист неопределенных слов (длины w) длины  t=2(r+1)
S[0] = P
for i = 1 to t-1 do:
   S[i] = S[i-1] + Q


# Основной цикл выдачи ключа
i = j = 0
A = B = 0
do 3 * max(t, c) times:
   A = S[i] = (S[i] + A + B) <<< 3
   B = L[j] = (L[j] + A + B) <<< (A + B)
   i = (i + 1) % t
   j = (j + 1) % c
```

Здесь (для w = 32):

$P = Odd((e – 2) * 2^w) = 0xB7E15163$

$Q = Odd((\phi – 2)*2^w) = 0x9E3779B9$

- «магические» константы. Функция Odd(x) возвращает ближайшее к *x* нечетное число.

```
# Функция шифровки,
# «<<<» - циклический побитовый сдвиг влево
# «^» - побитовое исключающее ИЛИ
A = A + S[0]
B = B + S[1]
for i = 1 to r do:
   A = ((A ^ B) <<< B) + S[2 * i]
   B = ((B ^ A) <<< A) + S[2 * i + 1]
```

```
return A, B
```

Здесь r – количество раундов. Рекомендуется: 12 – 20.

```
# Функция дешифровки. Прямолинейное обращение функции шифровки
for i = r down to 1 do:
   B = ((B - S[2 * i + 1]) >>> A) ^ A
   A = ((A - S[2 * i]) >>> B) ^ B
B = B - S[1]
A = A - S[0]
return A, B
```

2. Режимы блочного шифрования

Блок-шифр сам по себе позволяет скрыть запись только одного блока известной длины. Для сообщения переменной длины информация должна быть разделена на отдельные блоки шифрования. В простейшем случае, называемом режимом электронной кодовой книги (ECB), сообщение разбивается на отдельные блоки размера блока шифрования (с возможным расширением последнего блока при помощи амортизирующих битов), и каждый блок зашифровывается и дешифруется по отдельности (к нему применяется схема, изображенная на рис. 2).

Однако такая простая методология независимой шифровки блоков в целом небезопасна – одинаковые блоки открытого текста могут генерировать одинаковые блоки зашифрованного текста (для аналогичного ключа), поэтому паттерны внутри сообщения открытого текста становятся очевидными в выводе зашифрованного текста.

Для преодоления этого ограничения разработано множество режимов блочного шифрования [3]. Режим представляет из себя некий метод связывания блоков между собой, а также с некоторым случайным блоком, называемым вектором инициализации. Вектор инициализации обязательно должен быть случайным или псевдослучайным, чтобы не позволить злоумышленнику вывести соотношения между сегментами шифротекста, применяя схему шифрования повторно с одним и тем же ключом.

Далее рассмотрим 4 самых основных режима блочного шифрования: ECB (режим электронной кодовой книги), CBC (режим сцепления блоков шифротекста), PCBC (режим распространяющегося сцепления блоков шифра) и CFB (режим обратной связи по шифротексту).

2.1. Режим электронной кодовой книги (ECB)

Самый простой режим. В нем исходный текст разбивается на блоки, которые затем шифруются независимо друг от друга (рис. 2).

Недостатком этого метода является то, что идентичные блоки исходного текста зашифрованы в одинаковые блоки зашифрованного текста; таким образом, он не скрывает закономерности данных. В некотором смысле он не обеспечивает конфиденциальность сообщений, и его не рекомендуется использовать в криптографических протоколах вообще.

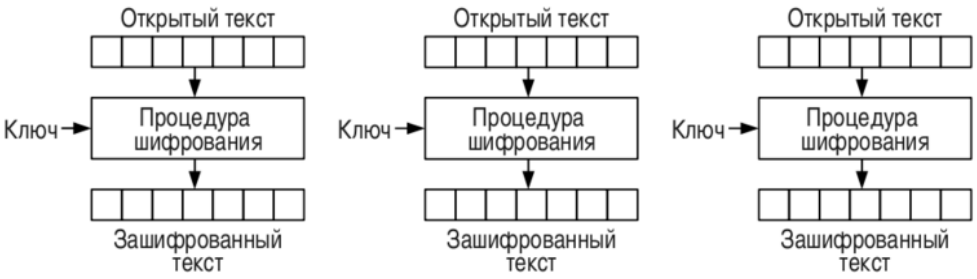

Рисунок 2 - Блок-схема режима ECB

2.2. Режим сцепления блоков шифротекста (CBC)

В режиме CBC каждый блок открытого текста объединяется посредством операции побитового исключающего ИЛИ с предыдущим блоком зашифрованного текста перед зашифровкой. Таким образом, каждый блок зашифрованного текста зависит от всех блоков открытого текста, обработанных до этой точки. Чтобы каждое сообщение было уникальным, в первом блоке должен использоваться вектор инициализации – чаще всего, случайно сгенерированная последовательность бит длиной в один блок.

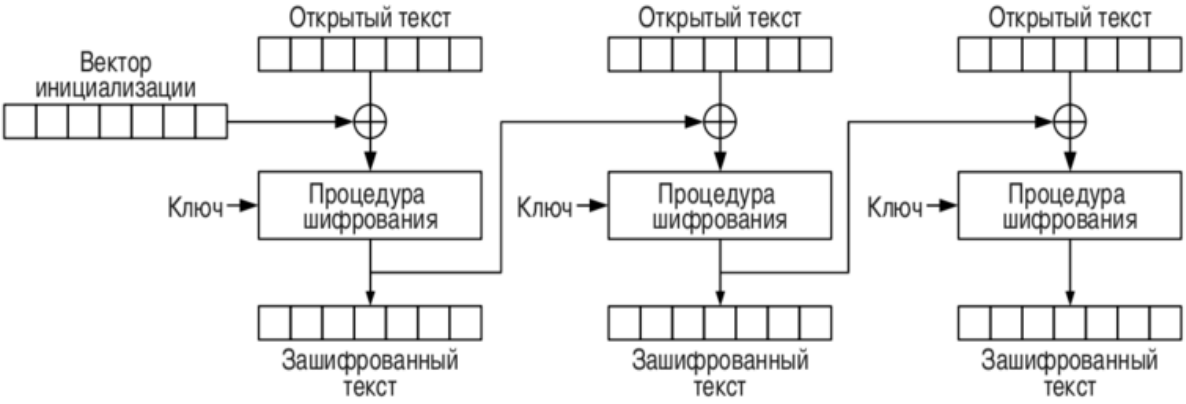

Рисунок 3 – Блок схема CBC

2.3. Режим распространяющегося сцепления блоков шифротекста

Этот режим был разработан для того, чтобы небольшие изменения в зашифрованном тексте распространялись неограниченно при расшифровке, а также при шифровании. В режиме PCBC каждый блок открытого текста объединяется при помощи исключающего ИЛИ как с предыдущим блоком незашифрованного текста, так и с предыдущим блоком зашифрованного текста перед зашифровкой. Как и в режиме CBC, в первом блоке используется вектор инициализации.

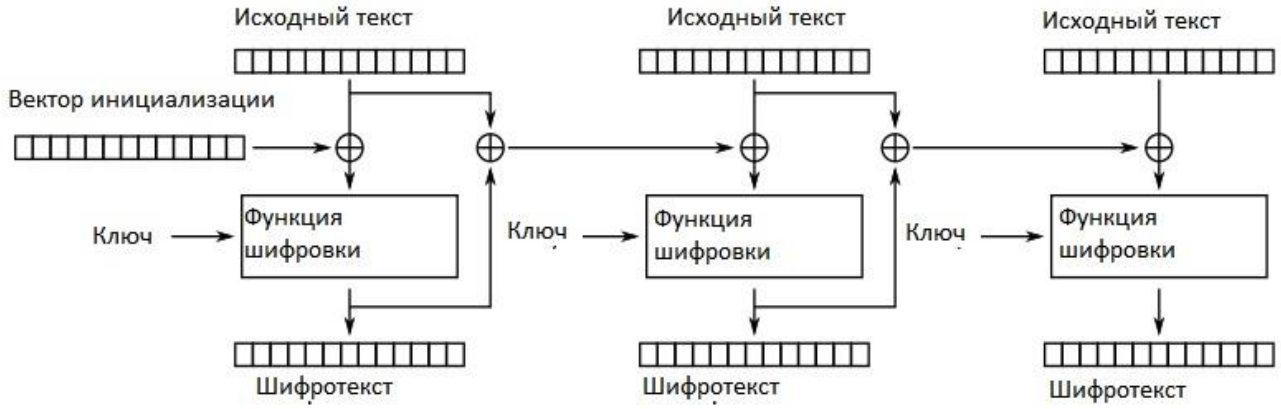

Рисунок 4 – Блок-схема режима PCBC

3. Практическая часть

В рамках данной работы была реализована программа на языке Python с применением фреймворка NumPy для осуществления математических операций с числами фиксированной длины в битах. Программа осуществляет шифрование/дешифрование файлов произвольного размера, произвольного содержания (двоичные) при помощи алгоритма RC5. Из режимов блочного шифрования реализованы: ECB, CBC, PCBC, CFB. Программа легко может быть расширена для применения практически любого алгоритма шифрования: нужно просто передать соответствующие объекты функций в основную функцию *process* (рис. 3).

```
def process(block_fn, opmode_fn, expand_fn, key, filename_in,
filename_out, block_struct="2I", strip_bytes=0,
            IV=None):
    """ block_fn: функция, [де]шифрующая отдельно взятый блок,
например rc5_encrypt/rc5_decrypt
        opmode_fn: функция, реализующая алгоритм блочного шифрования
        expand_fn: функция для расширения ключа. Если None, то ключ
не расширяется
        key: ключ
        filename_in: имя входного файла
        filename_out: имя выходного файла
        block_struct = "2I": Структура одного блока, подаваемого на
вход алгоритму.
            2 беззнаковых 4-байтовых целых числа по умолчанию
        strip_bytes = 0: сколько байт отбросить при записи выходного
файла. 0 по умолчанию
        IV: вектор инициализации
    """
```

Рисунок 3 - Сигнатура и описание параметров функции *process*

Далее приведен вывод программы (режим CFB). Для обозначения непечатных ASCII символов в Python применяется нотация \x{код символа}, код символа – шестнадцатиричное число 0 - 255:

1. Текстовый файл

```
=== ДО шифрования:
Привет, мир! Hello, world!
=== После шифрования:
6  байт добавлено
b'k\xfb.m\nS\x8c\x0e\xd9\x19[\xfaK\xe1\xd7@_z>4|\x1e\xf7o\x8e.h3\xd9K!\x88'
=== После дешифрования:
Привет, мир! Hello, world!
```

2. Изображение JPEG (выводятся первые 64 байта файла). Заметно, что сигнатура файла (“JFIF”) восстановлена после дешифровки.

```
=== ДО шифрования:
b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x02\x00\x00d\x00d\x00\x00\xff\xec\x00\x11Duck
y\x00\x01\x00\x04\x00\x00\x00P\x00\x00\xff\xee\x00\x0eAdobe\x00d\xc0\x00\x00\x00\x01
\xff\xdb\x00\x84\x00\x02\x02\x02\x02'... еще 19260 байт
=== После шифрования:
4  байт добавлено
b'\xdd\xd9\xa7\xac\x80s\n\xbb\xfbX\xc4\x9e`c\xbf<7\x93\xdf\xb9T\x00\x9a\x8f\xaa@\x0e
3\xd0\xe1f\xb0E\xb7\x1c\xfd\xea\xa1\xe4\xb2T<\xb6\x81\xd5d\xd9+\xe0\xe1L\x95a\x9b3\x
e9F\xa1\xd6\xaa\xb2\xfa\x04\x02'... еще 19264 байт
=== После дешифрования:
b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x02\x00\x00d\x00d\x00\x00\xff\xec\x00\x11Duck
y\x00\x01\x00\x04\x00\x00\x00P\x00\x00\xff\xee\x00\x0eAdobe\x00d\xc0\x00\x00\x00\x01
\xff\xdb\x00\x84\x00\x02\x02\x02\x02'... еще 19260 байт
```

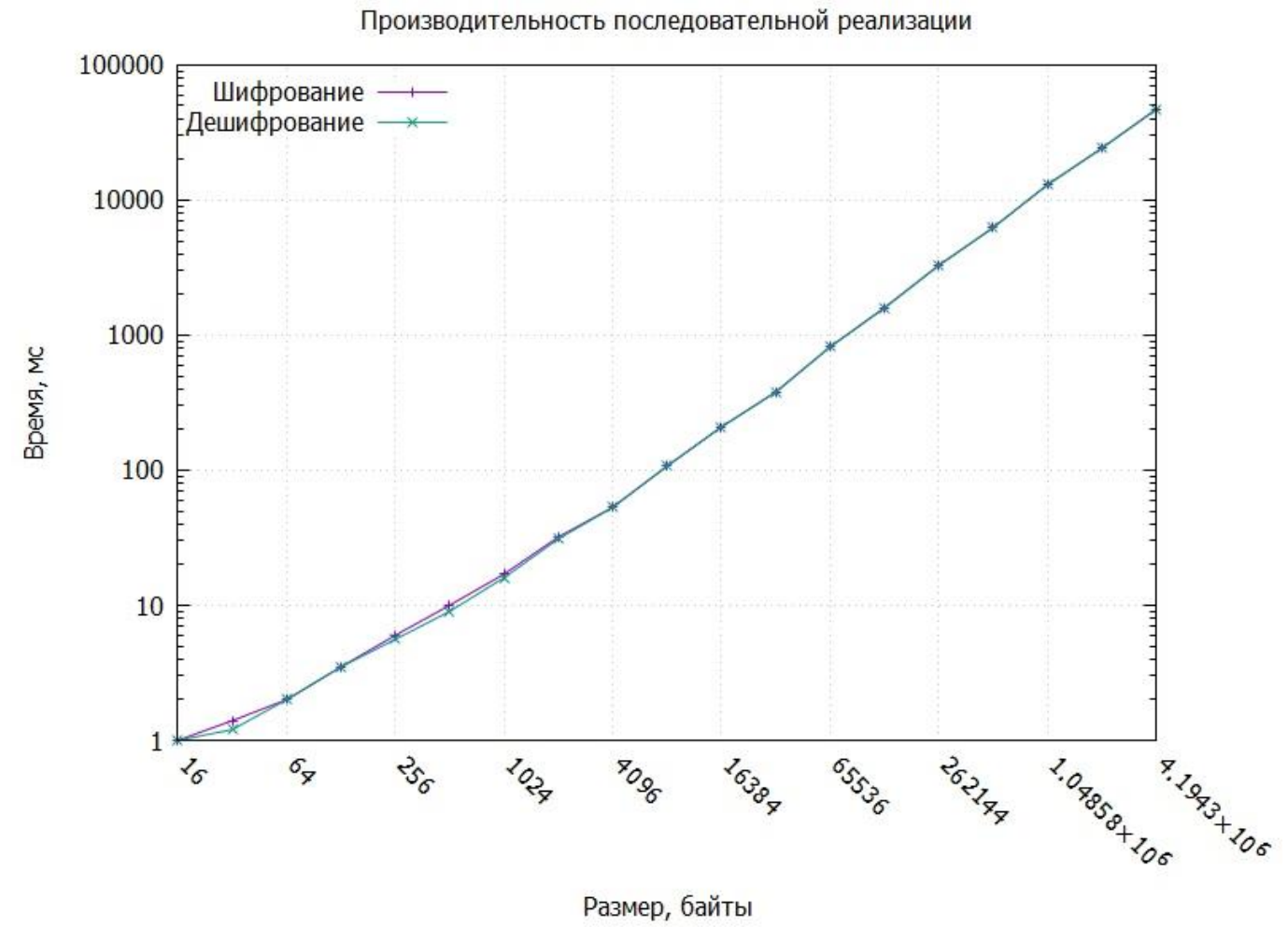

Рисунок 4 – Производительность последовательной реализации.

На рисунке 4 видно, что процедуры шифрования и дешифрования почти не различаются по временным затратам, поэтому далее будем рассматривать только процедуру шифрования.

Заключение

Был рассмотрены методы блочного шифрования, в частности алгоритм RC5, который был реализован последовательным (для режимов ECB, CBC, PCBC, CFB) и параллельным (для режима ECB) способом. Реализована программа на языке Python, осуществляющая шифрование файлов при помощи алгоритма RC5.

Было произведено сравнение времени выполнения последовательной программы на разных объёмах данных.

Список литературы

1. Applied Cryptography: Protocols, Algorithms, and Source Code in C, 2nd Edition. Bruce Schneier. ISBN: 978-0-471-11709-4.
2. Rivest, R. L. (1994). "The RC5 Encryption Algorithm"/ Proceedings of the Second International Workshop on Fast Software Encryption (FSE) 1994e. pp. 86–96.
3. NIST Computer Security Division's (CSD) Security Technology Group (STG). "Block cipher modes". Cryptographic Toolkit. NIST.
4. B. Moeller (May 20, 2004), Security of CBC Ciphersuites in SSL/TLS: Problems and Countermeasures