Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра информатики

Дисциплина: Методы защиты информации

ОТЧЁТ
к лабораторной работе №1
на тему

СИММЕТРИЧНАЯ КРИПТОГРАФИЯ. СТАНДАРТ ШИФРОВАНИЯ ГОСТ 28147-89

Выполнил: студент гр. 253503
Минич С.В.

Проверил: ассистент кафедры информатики Герчик А.В.

Минск 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Для практического изучения методов шифрования в рамках данной работы был выбран алгоритм симметричного блочного шифрования по ГОСТ 28147-89. ГОСТ 28147-89 представляет собой симметричный 64-битовый блочный алгоритм с 256-битовым ключом, предназначен для аппаратной и программной реализации, удовлетворяет криптографическим требованиям.

Алгоритм ГОСТ 28147-89, устаревший государственный стандарт СССР, является примером DES-подобных криптосистем, построенных на классической итерационной схеме Фейстеля и поддерживает несколько режимов работы для криптографического преобразования данных, таких как режим простой замены, когда все блоки данных шифруются независимо друг от друга. При этом для одинаковых блоков открытого сообщения будут получаться одинаковые блоки шифротекста. Режим гаммирования, где используется синхропосылка или вектор инициализации. Он подается на вход регистров, шифруется с использованием ключа, и результат подается снова на вход. Режим гаммирования с обратной связью: начинается со второго блока, используется результат шифрования предыдущего блока открытого текста [1].

Цель работы заключалась в изучении структуры алгоритма ГОСТ 28147-89 в режиме простой замены, реализации его основных компонентов и разработке программного обеспечения для шифрования и расшифровки текстовых данных. В ходе работы были подробно рассмотрены механизмы деления ключа, обработки блоков данных, применение преобразований и циклических сдвигов [2].

1 ХОД ВЫПОЛНЕНИЯ РАБОТЫ

В ходе выполнения работы была реализована криптографическая защита данных с использованием алгоритма ГОСТ 28147-89. Основная задача заключалась в том, чтобы обеспечить возможность шифрования и расшифрования текстовой информации с гарантией сохранения целостности. Алгоритм является блочным симметричным шифром, который оперирует 64-битными блоками данных и использует 256-битный ключ.

При реализации алгоритма данные преобразуются из байтов в 32-битные слова, что позволяет выполнять сложные побитовые и арифметические операции, определенные стандартом. Основу алгоритма составляет сеть Фейстеля, в которой каждый раунд преобразования состоит из следующих операций:

- сложение по модулю 2^32 левой части блока с раундовым подключом;
- нелинейное преобразование с использованием восьми 4-битных таблиц замен (*S*-блоки);
- циклический сдвиг полученного 32-битного слова на 11 бит влево;
- сложение по модулю 2 (*XOR*) результата с правой частью блока;
- перестановка правой и левой частей блока.

Этот процесс повторяется 32 раза для увеличения стойкости шифра, блок схема алгоритма представлена на рис. 1.

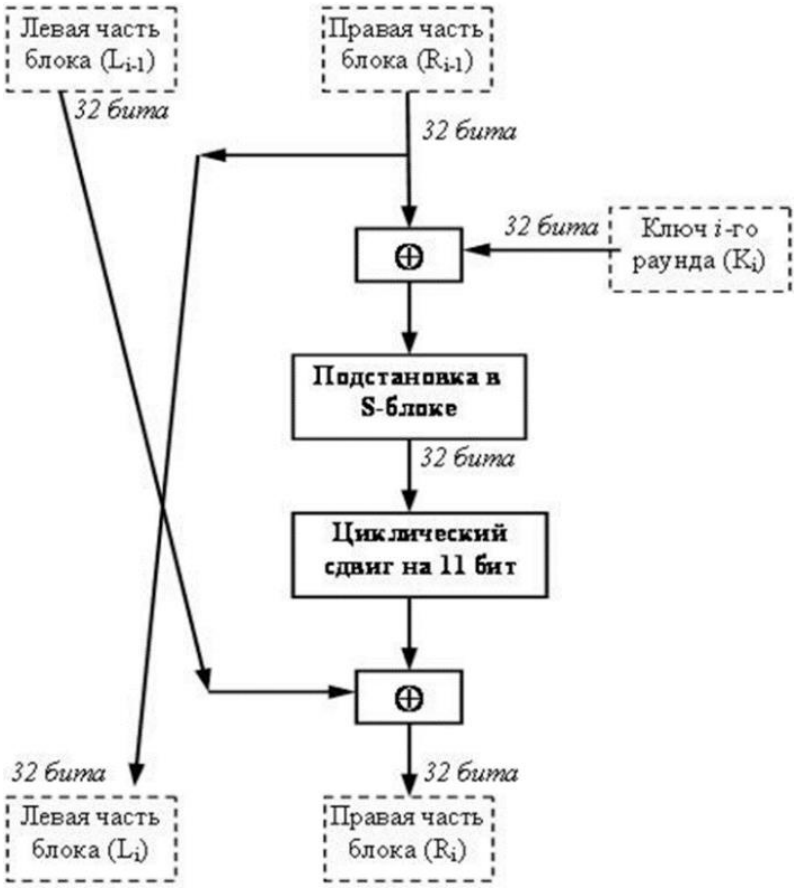

Рисунок 1 – Блок-схема одного раунда

Данная реализация включает режим простой замены, где каждый блок данных шифруется независимо. Это упрощает реализацию. В режиме простой замены каждый блок данных обрабатывается отдельно, после чего все зашифрованные блоки объединяются. Для расшифрования используется обратный порядок операций.

Код реализует блочный шифр согласно стандарту ГОСТ 28147-89. Центральным элементом является функция *gost_block_crypt*, которая инкапсулирует 32 раунда шифрования одного 64-битного блока. Для преобразования байтов в целые числа и обратно используются функции *bytes_to_int* и *int_to_bytes*, что обеспечивает точное соответствие арифметических операций требованиям стандарта. Внутреннее нелинейное преобразование блока реализовано через таблицы замены *S_BLOCKS* и циклические сдвиги, что создает криптографическую стойкость и усложняет анализ зашифрованных данных. Для корректной работы с данными произвольной длины реализована функция дополнения *padding*. Пример шифрования-расшифрования представлен на рис. 2.

```
Original text: Test message

Simple Substitution
Decrypted text: Test message
```

Рисунок 2 – Пример выполнения шифрования-расшифрования

ЗАКЛЮЧЕНИЕ

В ходе работы был подробно изучен и реализован алгоритм блочного шифрования, определённый стандартом ГОСТ 28147-89. На основе теоретических положений стандарта была создана программа на *Python*, включающая как базовые операции над блоками данных, так и применение различных режимов шифрования, а именно простой подстановки. Реализация позволила продемонстрировать работу ключевых элементов алгоритма: использование таблицы *S*-блоков, циклических сдвигов и построение раундовых ключей.

Полученные результаты подтвердили корректность подхода и показали, что алгоритм может быть воспроизведён средствами общего назначения с сохранением криптографических свойств. Программа обеспечивает как шифрование, так и последующее успешное расшифрование информации, что подтверждает правильность реализации. Таким образом, цель работы была достигнута: изучение стандарта и его программное воплощение позволили на практике закрепить понимание теоретических основ современной криптографии и методов защиты информации.

.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Системы обработки информации. Защита криптографическая. Алгоритмы криптографического преобразования [Электронный ресурс]. – Режим доступа: https://meganorm.ru/Data2/1/4294826/4294826631.pdf – Дата доступа: 15.09.2025

[2] Алгоритм шифрования ГОСТ 28147-89. Метод простой замены. [Электронный ресурс]. – Режим доступа: https://wasm.in/blogs/algoritm-shifrovanija-gost-28147-89-metod-prostoj-zameny.359/ – Дата доступа: 15.09.2025

ЛИСТИНГ ПРОГРАММНОГО КОДА

```
UINT32_MAX = 0xFFFFFFFF # 2^32 - 1

# S block = 4 bit
S_BLOCKS = [
    [4, 10, 9, 2, 13, 8, 0, 14, 6, 11, 1, 12, 7, 15, 5, 3],
    [14, 11, 4, 12, 6, 13, 15, 10, 2, 3, 8, 1, 0, 7, 5, 9],
    [5, 8, 1, 13, 10, 3, 4, 2, 14, 15, 12, 7, 6, 0, 9, 11],
    [7, 13, 10, 1, 0, 8, 9, 15, 14, 4, 6, 12, 11, 2, 5, 3],
    [6, 12, 7, 1, 5, 15, 13, 8, 4, 10, 9, 14, 0, 3, 11, 2],
    [4, 11, 10, 0, 7, 2, 1, 13, 3, 6, 8, 5, 9, 12, 15, 14],
    [13, 11, 4, 1, 3, 15, 5, 9, 0, 10, 14, 7, 6, 8, 2, 12],
    [1, 15, 13, 0, 5, 7, 10, 4, 9, 2, 3, 14, 6, 11, 8, 12]
]


def add_mod_2_32(block, subkey):
    return (block + subkey) & UINT32_MAX


def s_block_substitution(value):
    result = 0
    # every 4 bits in half-block swap with S_BLOCKS number
    for i in range(8):
        block = (value >> (i * 4)) & 0xF
        substituted_block = S_BLOCKS[i][block]
        result |= (substituted_block << (i * 4))
    return result


def rotate_left(value):
    return ((value << 11) & UINT32_MAX) | (value >> (32 - 11))


def generate_subkeys(key_256_bit):
    subkeys = []
    # divide into 32-bit
    for i in range(8):
        subkey = (key_256_bit >> (256 - (i + 1) * 32)) & UINT32_MAX
        subkeys.append(subkey)

    encryption_subkeys = []
    for _ in range(3):
        encryption_subkeys.extend(subkeys)
    encryption_subkeys.extend(list(reversed(subkeys)))

    decryption_subkeys = list(reversed(encryption_subkeys))

    return encryption_subkeys, decryption_subkeys


def gost_block_crypt(block_64_bit, subkeys):
    A = (block_64_bit >> 32) & UINT32_MAX  # left
    B = block_64_bit & UINT32_MAX          # right

    for i in range(32):
        f = add_mod_2_32(B, subkeys[i])
        f = s_block_substitution(f)
        f = rotate_left(f)
        new_A = f ^ A    # sum mod 2 == XOR

        A = B
        B = new_A

    # ?
    final_A = B
    final_B = A

    return (final_A << 32) | final_B


def bytes_to_int(bytes_data):
    return int.from_bytes(bytes_data, 'big')

def int_to_bytes(int_data, length):
    return int_data.to_bytes(length, 'big')

def padding(data, block_size=8):
    padding_len = block_size - (len(data) % block_size)
    padding = bytes([padding_len]) * padding_len
    return data + padding, padding_len

def unpadding(data, padding_len):
    return data[:-padding_len]


def gost_simple_substitution(data, subkeys, padding_len=None, encrypt=True):
    block_size = 8
```

```
if encrypt:
    data, padding_len = padding(data, block_size)

# divide data into 64-bit blocks
processed_blocks = []
for i in range(0, len(data), block_size):
    block_bytes = data[i:i + block_size]
    block_int = bytes_to_int(block_bytes)

    processed_block_int = gost_block_crypt(block_int, subkeys)
    processed_block_bytes = int_to_bytes(processed_block_int, block_size)
    processed_blocks.append(processed_block_bytes)

result = b''.join(processed_blocks)

if not encrypt:
    result = unpadding(result, padding_len)

return result, padding_len if encrypt else None
```