Белорусский государственный университет информатики и радиоэлектроники

Кафедра информатики

Лабораторная работа № 2

Симметричная криптография. СТБ 34.101.31-2011.

Выполнила студентка гр. 653502: Сулима М.Ф.

Проверил ассистент КИ: Артемьев В. С.

Минск, 2019

Введение

Настоящий стандарт определяет семейство криптографических алгоритмов, предназначенных для обеспечения конфиденциальности и контроля целостности данных. Обрабатываемыми данными являются двоичные слова (сообщения).

Криптографические алгоритмы стандарта построены на основе базовых алгоритмов шифрования блока данных. Криптографические алгоритмы шифрования и контроля целостности делятся на восемь групп:

1) алгоритмы шифрования в режиме простой замены;
2) алгоритмы шифрования в режиме сцепления блоков;
3) алгоритмы шифрования в режиме гаммирования с обратной связью;
4) алгоритмы шифрования в режиме счетчика;
5) алгоритм выработки имитовставки;
6) алгоритмы одновременного шифрования и имитозащиты данных;
7) алгоритмы одновременного шифрования и имитозащиты ключа;
8) алгоритм хэширования.

Первые четыре группы предназначены для обеспечения конфиденциальности сообщений. Каждая группа включает алгоритм зашифрования и алгоритм расшифрования. Стороны, располагающие общим ключом, могут организовать конфиденциальный обмен сообщениями путем их шифрования перед отправкой и расшифрования после получения. В режимах простой замены и сцепления блоков шифруются сообщения, которые содержат хотя бы один блок, а в режимах гаммирования с обратной связью и счетчика — сообщения произвольной длины.

В рамках лабораторной работы необходимо реализовать программные средства шифрования и дешифрования при помощи алгоритма СТБ 34.101.31-2011 в различных режимах.

Блок-схема алгоритма

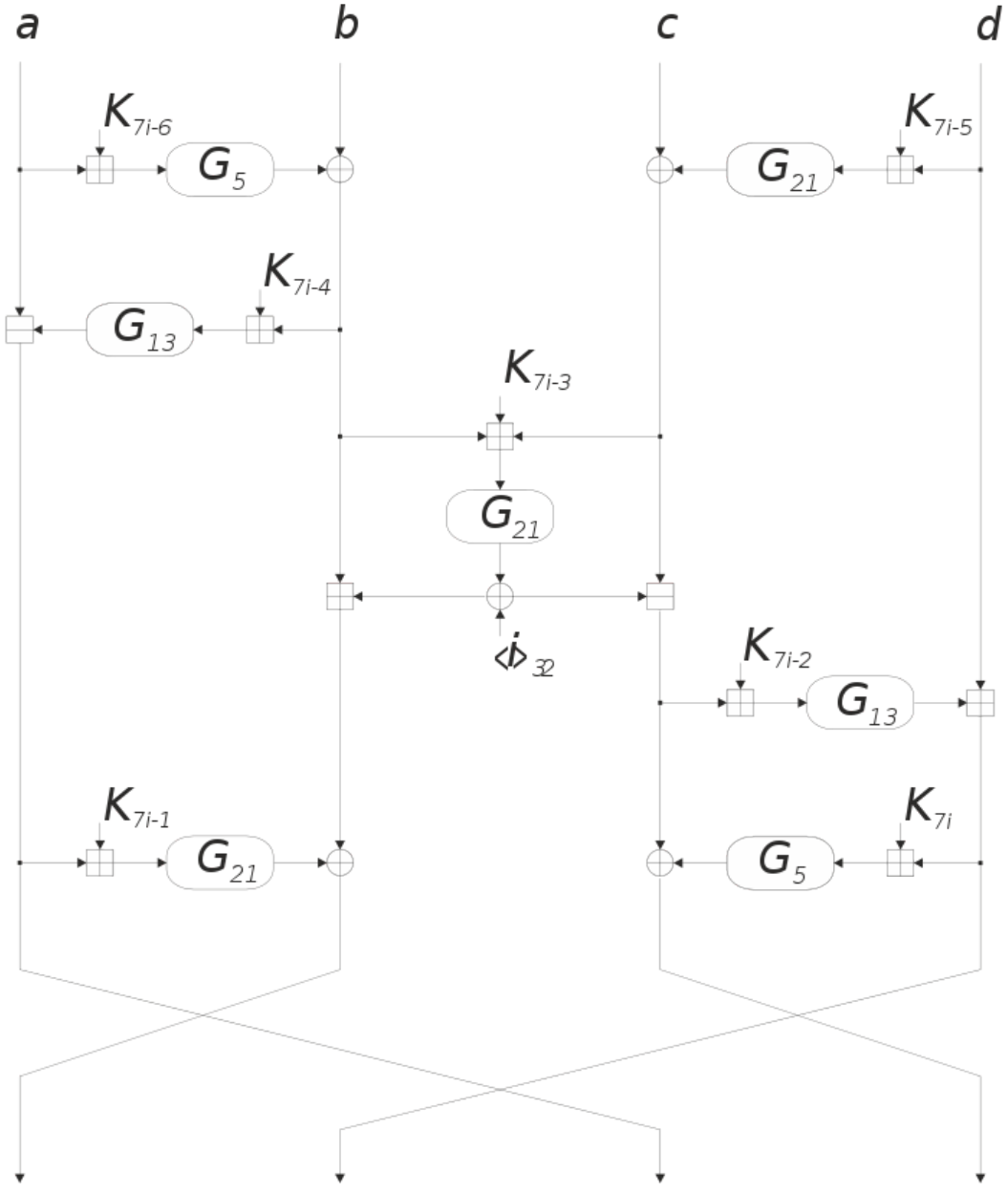

Рис.1. Вычисления на i-ом такте шифрования.

Пример выполнения программы

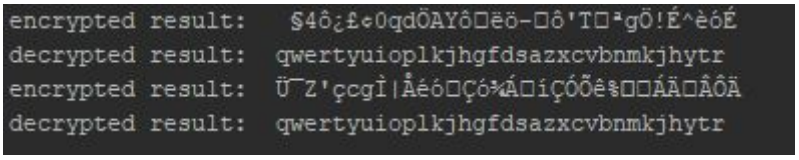

Рис. 2. Пример работы программы

Код программы

```
def encrypt(self, text):
   bits_text = str_to_bit_array(text)  # 16 байт (блок) -> 128
бит
   words = split_lists(bits_text, 32)
   a = words[0]
   b = words[1]
   c = words[2]
   d = words[3]
   for i in range(1, 9):
       sub_key = copy.deepcopy(self.sub_keys[7 * i - 6 - 1])
       value = plus_mod_32(a, sub_key)
       value = self.G_substitution(value, 5)
       b = [k ^ l for k, l in zip(b, value)]
       sub_key = copy.deepcopy(self.sub_keys[7 * i - 5 - 1])
       value = plus_mod_32(d, sub_key)
       value = self.G_substitution(value, 21)
       c = [k ^ l for k, l in zip(c, value)]
       sub_key = copy.deepcopy(self.sub_keys[7 * i - 4 - 1])
       value = plus_mod_32(b, sub_key)
       value = self.G_substitution(value, 13)
       a = minus_mod_32(a, value)
       sub_key = copy.deepcopy(self.sub_keys[7 * i - 3 - 1])
       value = plus_mod_32(b, c)
       value = plus_mod_32(value, sub_key)
       value = self.G_substitution(value, 21)
       value_i = i % (2 ** 32)
       value_i = int_to_bit_array(value_i, 32)
       while len(value_i) < 32:
           value_i.insert(0, 0)
       e = [k ^ l for k, l in zip(value, value_i)]
       b = plus_mod_32(b, e)
       c = minus_mod_32(c, e)
       sub_key = copy.deepcopy(self.sub_keys[7 * i - 2 - 1])
       value = plus_mod_32(c, sub_key)
       value = self.G_substitution(value, 13)
       d = plus_mod_32(d, value)
       sub_key = copy.deepcopy(self.sub_keys[7 * i - 1 - 1])
       value = plus_mod_32(a, sub_key)
       value = self.G_substitution(value, 21)
       b = [k ^ l for k, l in zip(b, value)]
       sub_key = copy.deepcopy(self.sub_keys[7 * i - 1])
```

```
            value = plus_mod_32(d, sub_key)
            value = self.G_substitution(value, 5)
            c = [k ^ l for k, l in zip(c, value)]
            a, b = b, a
            c, d = d, c
            b, c = c, b
        encrypted = b + d + a + c
        encrypted = bit_array_to_str(encrypted)
        return encrypted

    def decrypt(self, text):
        bits_text = str_to_bit_array(text)
        words = split_lists(bits_text, 32)
        a = words[0]
        b = words[1]
        c = words[2]
        d = words[3]
        for i in range(8, 0, -1):
            sub_key = copy.deepcopy(self.sub_keys[7 * i - 1])
            temp = plus_mod_32(a, sub_key)
            temp = self.G_substitution(temp, 5)
            b = [k ^ l for k, l in zip(b, temp)]
            sub_key = copy.deepcopy(self.sub_keys[7 * i - 1 - 1])
            temp = plus_mod_32(d, sub_key)
            temp = self.G_substitution(temp, 21)
            c = [k ^ l for k, l in zip(c, temp)]
            sub_key = copy.deepcopy(self.sub_keys[7 * i - 2 - 1])
            temp = plus_mod_32(b, sub_key)
            temp = self.G_substitution(temp, 13)
            a = minus_mod_32(a, temp)
            sub_key = copy.deepcopy(self.sub_keys[7 * i - 3 - 1])
            temp = plus_mod_32(b, c)
            temp = plus_mod_32(temp, sub_key)
            temp = self.G_substitution(temp, 21)
            value_i = i % (2 ** 32)
            value_i = int_to_bit_array(value_i, 32)
            e = [k ^ l for k, l in zip(temp, value_i)]
            b = plus_mod_32(b, e)
            c = minus_mod_32(c, e)
            sub_key = copy.deepcopy(self.sub_keys[7 * i - 4 - 1])
            temp = plus_mod_32(c, sub_key)
            temp = self.G_substitution(temp, 13)
```

```
        d = plus_mod_32(d, temp)
        sub_key = copy.deepcopy(self.sub_keys[7 * i - 5 - 1])
        temp = plus_mod_32(a, sub_key)
        temp = self.G_substitution(temp, 21)
        b = [k ^ l for k, l in zip(b, temp)]
        sub_key = copy.deepcopy(self.sub_keys[7 * i - 6 - 1])
        temp = plus_mod_32(d, sub_key)
        temp = self.G_substitution(temp, 5)
        c = [k ^ l for k, l in zip(c, temp)]
        a, b = b, a
        c, d = d, c
        a, d = d, a
    decoded = c + a + d + b
    decoded = bit_array_to_str(decoded)
    return decoded
```

Вывод

Настоящий стандарт определяет семейство криптографических алгоритмов шифрования и контроля целостности, которые используются для защиты информации при ее хранении, передаче и обработке. Настоящий стандарт применяется при разработке средств криптографической защиты информации.