Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

КАФЕДРА ИНФОРМАТИКИ

Отчёт по лабораторной работе №2

По теме «Разработка криптографических алгоритмов шифрования из СТБ П 34.101.31-2007»

Выполнил:

студент гр. 853503

Климович А.А

Проверил:

Протько М.И

Оглавление

[Цель работы 3](#_Toc84278617)

[Описание реализуемых алгоритмов 4](#_Toc84278618)

[1. Шифрование блоков данных 4](#_Toc84278619)

[2. Шифрование в режиме простой замены 5](#_Toc84278620)

[3. Шифрование в режиме сцепления блоков 6](#_Toc84278621)

[4. Шифрование в режиме гаммирования с обратной связью 7](#_Toc84278622)

[5. Шифрование в режиме счетчика 8](#_Toc84278623)

[Результаты работы 9](#_Toc84278624)

[Вывод 10](#_Toc84278625)

[Приложение 1. Программный код 11](#_Toc84278626)

# Цель работы

В ходе выполнения лабораторной работы необходимо реализовать следующие методы шифрования: шифрование в режиме простой замены, шифрование в режиме сцепления блоков, шифрование в режиме гаммирования с обратной связью и шифрование в режиме счетчика, алгоритмы которых описаны в предварительном государственном стандарте Республики Беларуси СТБ П 34.101.31-2007.

# Описание реализуемых алгоритмов

## Шифрование блоков данных

**Входные и выходные данные**

Входными данными алгоритмов зашифрования и расшифрования являются сло- во X ∈ {0,1}128 и ключ θ ∈ {0,1}256.

Выходными данными является слово Y ∈ {0, 1}128 , результат зашифрования либо расшифрования слова X на ключе θ: Y = Fθ(X) либо Y = F−1(X).

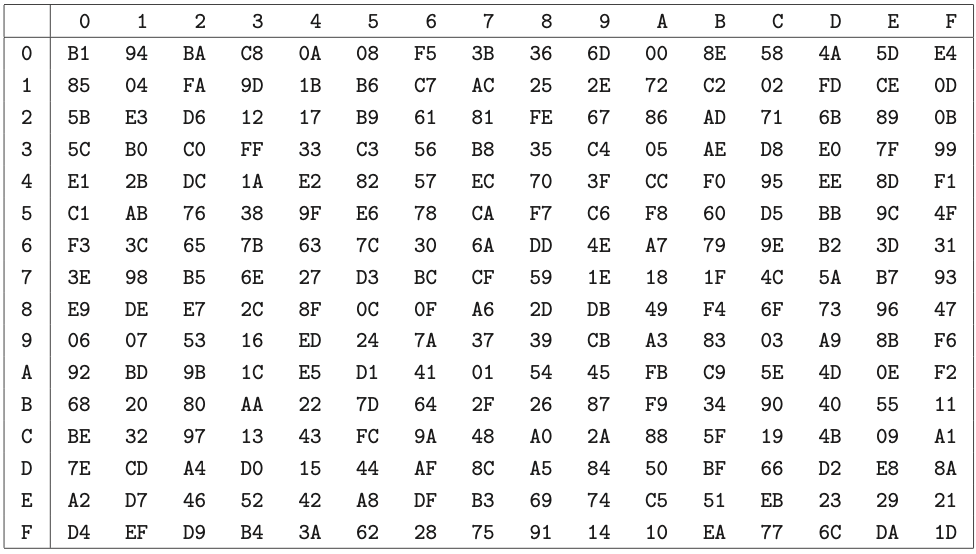
Входные данные подготавливаются следующим образом:  
1. Слово X записывается в виде X = X1 ∥ X2 ∥ X3 ∥ X4, где Xi ∈ {0,1}32.  
2. Ключ θ записывается в виде θ = θ1 ∥ θ2 ∥ ... ∥ θ8, θi ∈ {0,1}32, и определяются тактовые ключи K1 = θ1,K2 = θ2,...,K8 = θ8,K9 = θ1,K10 = θ2,...,K56 = θ8.

**Переменные и вспомогательные преобразования**

*Переменные.* Используются переменные a, b, c, d, e со значениями из {0, 1}32.

*Подстановка H .* Подстановка H : {0, 1}8 → {0, 1}8 задается таблицей 1. В таблице 1 используется шестнадцатеричное представление слов u ∈ {0, 1}8. Если u = IJ16, то значе- ние H(u) находится на пересечении строки I и столбца J. Например, H(A216) = 9B16.

Таблица 1 — Подстановка H



*Преобразования Gr (r = 5, 13, 21).* Преобразование Gr : {0, 1}32 → {0, 1}32 ставит в соответствие слову u = u1 ∥ u2 ∥ u3 ∥ u4, ui ∈ {0, 1}8, слово

Gr(u) = λr (H(u1) ∥ H(u2) ∥ H(u3) ∥ H(u4)).

**Алгоритм зашифрования**

Для зашифрования слова X на ключе θ выполняются следующие шаги:

1. Установитьa←X1,b←X2,c←X3,d←X4.
2. Для i = 1,2,...,8 выполнить (см. рис. 1):

1) b←b⊕G5(a􏰁K7i−6);  
2) c←c⊕G21(d􏰁K7i−5);  
3) a←a􏰂G13(b􏰁K7i−4);  
4) e←G21(b􏰁c􏰁K7i−3)⊕⟨i⟩32; 5) b←b􏰁e;

6) c←c􏰂e;  
7) d←d􏰁G13(c􏰁K7i−2); 8) b←b⊕G21(a􏰁K7i−1); 9) c←c⊕G5(d􏰁K7i);

10) a↔b; 11) c↔d; 12) b↔c.

1. УстановитьY ←b∥d∥a∥c.
2. Возвратить Y .

**Алгоритм расшифрования**

Для расшифрования слова X на ключе θ выполняются следующие шаги:

1. Установитьa←X1,b←X2,c←X3,d←X4.
2. Для i = 8,7,...,1 выполнить:

1) b←b⊕G5(a􏰁K7i);  
2) c←c⊕G21(d􏰁K7i−1);  
3) a←a􏰂G13(b􏰁K7i−2);  
4) e←G21(b􏰁c􏰁K7i−3)⊕⟨i⟩32; 5) b←b􏰁e;  
6) c←c􏰂e;  
7) d←d􏰁G13(c􏰁K7i−4);  
8) b←b⊕G21(a􏰁K7i−5);  
9) c←c⊕G5(d􏰁K7i−6);

10) a↔b; 11) c↔d; 12) a↔d.

1. УстановитьY ←c∥a∥d∥b.
2. Возвратить Y .

## Шифрование в режиме простой замены

**Входные и выходные данные**

Входными данными алгоритмов зашифрования и расшифрования являются слово X∈{0,1}128∗ и ключθ∈{0,1}256.

Выходными данными является слово Y ∈ {0,1}|X| — результат зашифрования либо расшифрования X на ключе θ.

Входное слово X записывается в виде  
X =X1 ∥X2 ∥...∥Xd, Xi ∈{0,1}128.

При шифровании словам Xi ставятся в соответствие слова Yi ∈ {0, 1}128, из которых затем составляется Y .

**Алгоритм зашифрования**

Зашифрование слова X на ключе θ состоит в выполнении следующих шагов:

1. Для i = 1,2,...,d выполнить Yi ← Fθ(Xi).
2. УстановитьY ←Y1 ∥Y2 ∥...∥Yd.
3. Возвратить Y .

**Алгоритм расшифрования**

Расшифрование слова X на ключе θ состоит в выполнении следующих шагов:

1. Для i = 1,2,...,d выполнить Yi ← F−1(Xi).
2. УстановитьY ←Y1 ∥Y2 ∥...∥Yd.
3. Возвратить Y .

## Шифрование в режиме сцепления блоков

**Входные и выходные данные**

Входными данными алгоритмов зашифрования и расшифрования являются слово X ∈ {0, 1}128∗, ключ θ ∈ {0, 1}256 и синхропосылка S ∈ {0, 1}128.

Выходными данными является слово Y ∈ {0,1}|X| — результат зашифрования либо расшифрования X на ключе θ при использовании синхропосылки S.

Входное слово X записывается в виде  
X =X1 ∥X2 ∥...∥Xd, Xi ∈{0,1}128.

При шифровании словам Xi ставятся в соответствие слова Yi ∈ {0, 1}128, из которых затем составляется Y .

**Алгоритм зашифрования**

Зашифрование слова X на ключе θ при использовании синхропосылки S состоит в выполнении следующих шагов:

1. Для i=1,2,...,dвыполнитьYi ←Fθ(Xi⊕Yi−1),гдеY0 =Fθ(S).
2. УстановитьY ←Y1 ∥Y2 ∥...∥Yd.
3. Возвратить Y .

**Алгоритм расшифрования**

Расшифрование слова X на ключе θ при использовании синхропосылки S состоит в выполнении следующих шагов:

1. Для i=1,2,...,dвыполнитьYi ←F−1(Xi)⊕Xi−1,гдеX0 =Fθ(S). θ
2. УстановитьY ←Y1 ∥Y2 ∥...∥Yd.
3. Возвратить Y .

## Шифрование в режиме гаммирования с обратной связью

**Входные и выходные данные**

Входными данными алгоритмов зашифрования и расшифрования являются слово X ∈ {0, 1}∗, ключ θ ∈ {0, 1}256 и синхропосылка S ∈ {0, 1}128.

Выходными данными является слово Y ∈ {0,1}|X| — результат зашифрования либо расшифрования X на ключе θ при использовании синхропосылки S.

Входное слово X записывается в виде  
X =X1 ∥X2 ∥...∥Xd, |X1|=|X2|=...=|Xd−1|=128, |Xd|≤128.

При шифровании словам Xi ставятся в соответствие слова Yi ∈ {0,1}|Xi|, из которых затем составляется Y .

**Алгоритм зашифрования**

Зашифрование слова X на ключе θ при использовании синхропосылки S состоит в выполнении следующих шагов:

1. Для i=1,2,...,d выполнить Yi ←Xi ⊕L|Xi|(Fθ(Yi−1)),гдеY0 =S.
2. УстановитьY ←Y1 ∥Y2 ∥...∥Yd.
3. Возвратить Y .

**Алгоритм расшифрования**

Расшифрование слова X на ключе θ при использовании синхропосылки S состоит в выполнении следующих шагов:

1. Для i=1,2,...,d выполнить Yi ←Xi⊕L|Xi|(Fθ(Xi−1)), где X0 =S.
2. УстановитьY ←Y1 ∥Y2 ∥...∥Yd.
3. Возвратить Y .

## Шифрование в режиме счетчика

**Входные и выходные данные**

Входными данными алгоритмов зашифрования и расшифрования являются слово X ∈ {0, 1}∗, ключ θ ∈ {0, 1}256 и синхропосылка S ∈ {0, 1}128.

Выходными данными является слово Y ∈ {0,1}|X| — результат зашифрования либо расшифрования X на ключе θ при использовании синхропосылки S.

Входное слово X записывается в виде  
X =X1 ∥X2 ∥...∥Xd, |X1|=|X2|=...=|Xd−1|=128, |Xd|≤128.

При шифровании словам Xi ставятся в соответствие слова Yi ∈ {0,1}|Xi|, из которых затем составляется Y .

**Переменные**

Используется переменная s со значениями из {0,1}128.

**Алгоритм зашифрования**

Зашифрование слова X на ключе θ при использовании синхропосылки S состоит в выполнении следующих шагов:

1. Установить s ← Fθ(S).  
2. Для i = 1,2,...,d выполнить:

1) s←s􏰁⟨1⟩128,

2) Yi ← Xi ⊕ L|Xi|(Fθ(s)).  
3. УстановитьY ←Y1 ∥Y2 ∥...∥Yd.

4. Возвратить Y .

**Алгоритм расшифрования**

Расшифрование слова X на ключе θ при использовании синхропосылки S состоит в выполнении тех же шагов, что и при зашифровании.

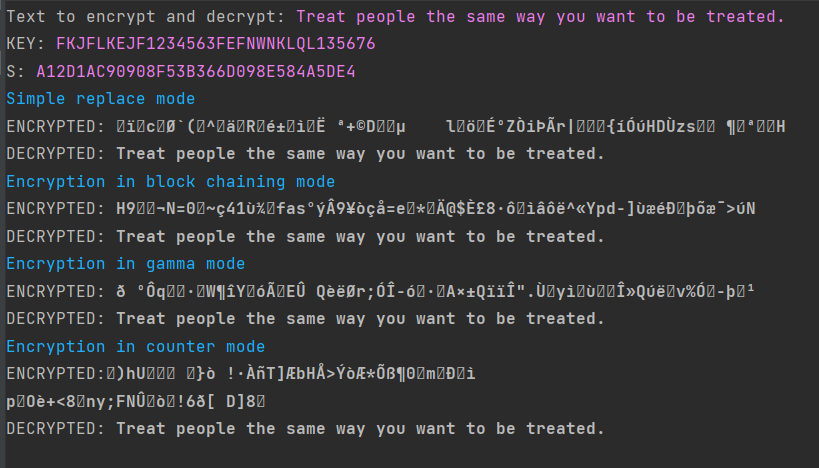
# Результаты работы

В ходе выполнения лабораторной работы были реализованы указанные выше методы шифрования.

Для демонстрации правильности работы алгоритмов были произведены операции шифрования/расшифровывания всеми реализованными алгоритмами со следующими входными данными:

* Шифруемая строка: Andrei wants to get a good points for Lab
* Значение ключа: FKJFLKEJF1234563FEFNWNKLQL135676
* Имитовставка s: A12D1AC90908F53B366D098E584A5DE4

Результаты работы:



Как видно из результатов работы, все реализованные методы успешно произвели операции шифрования и расшифрования, причем расшифрованная строка во всех случаях равна изначальной строке.

# Вывод

Таким образом, в ходе выполнения лабораторной работы были реализованы следующие методы шифрования: шифрование в режиме простой замены, шифрование в режиме сцепления блоков, шифрование в режиме гаммирования с обратной связью и шифрование в режиме счетчика.

После реализации все алгоритмы были успешно проверены на тестовых данных, в результате чего фактические результаты соответствуют ожидаемым.

# Приложение 1. Программный код

class STB:

def \_\_init\_\_(self):

self.H = [

[177, 148, 186, 200, 10, 8, 245, 59, 54, 109, 0, 142, 88, 74, 93, 228],

[133, 4, 250, 157, 27, 182, 199, 172, 37, 46, 114, 194, 2, 253, 206, 13],

[91, 227, 214, 18, 23, 185, 97, 129, 254, 103, 134, 173, 113, 107, 137, 11],

[92, 176, 192, 255, 51, 195, 86, 184, 53, 196, 5, 174, 216, 224, 127, 153],

[225, 43, 220, 26, 226, 130, 87, 236, 112, 63, 204, 240, 149, 238, 141, 241],

[193, 171, 118, 56, 159, 230, 120, 202, 247, 198, 248, 96, 213, 187, 156, 79],

[243, 60, 101, 123, 99, 124, 48, 106, 221, 78, 167, 121, 158, 178, 61, 49],

[62, 152, 181, 110, 39, 211, 188, 207, 89, 30, 24, 31, 76, 90, 183, 147],

[233, 222, 231, 44, 143, 12, 15, 166, 45, 219, 73, 244, 111, 115, 150, 71],

[6, 7, 83, 22, 237, 36, 122, 55, 57, 203, 163, 131, 3, 169, 139, 246],

[146, 189, 155, 28, 229, 209, 65, 1, 84, 69, 251, 201, 94, 77, 14, 242],

[104, 32, 128, 170, 34, 125, 100, 47, 38, 135, 249, 52, 144, 64, 85, 17],

[190, 50, 151, 19, 67, 252, 154, 72, 160, 42, 136, 95, 25, 75, 9, 161],

[126, 205, 164, 208, 21, 68, 175, 140, 165, 132, 80, 191, 102, 210, 232, 138],

[162, 215, 70, 82, 66, 168, 223, 179, 105, 116, 197, 81, 235, 35, 41, 33],

[212, 239, 217, 180, 58, 98, 40, 117, 145, 20, 16, 234, 119, 108, 218, 29]

]

@staticmethod

def add\_zeros(data, k):

if len(data) <= k:

zeros\_size = k - len(data)

data2 = [0 for i in range(zeros\_size)] + data

return data2

elif len(data) > k:

raise RuntimeError('{}, {}'.format(len(data), k))

@staticmethod

def get\_8\_keys(key):

if len(key) != 256:

raise RuntimeError('Key size is {} not 256'.format(len(key)))

keys = []

for i in range(0, 256, 32):

keys.append(key[i:i + 32])

return keys

def encrypt(self, data, key):

if type(data) != list:

raise RuntimeError('Not list')

key = self.add\_zeros(key, 256)

data = [1] + data

n = len(data)

m = 0

while m < n:

m += 128

data = self.add\_zeros(data, m)

res = []

for i in range(0, m, 128):

res += self.encrypt\_128(data[i:i+128], key)

return res

def encrypt\_128(self, data, key):

x = []

for i in range(0, 128, 32):

x.append(data[i:i+32])

a, b, c, d = x[0], x[1], x[2], x[3]

keys = self.get\_8\_keys(key)

tact\_keys = []

for i in range(7):

tact\_keys += keys

for i in range(8):

ak = (self.to\_int(a) + self.to\_int(tact\_keys[7 \* (i + 1) - 6 - 1])) % (2 \*\* 32)

ak = self.g(self.add\_zeros(self.to\_list(ak), 32), 5)

for j in range(32):

b[j] = b[j] ^ ak[j]

# 2) step 2

dk = (self.to\_int(d) + self.to\_int(tact\_keys[7 \* (i + 1) - 5 - 1])) % (2 \*\* 32)

dk = self.g(self.add\_zeros(self.to\_list(dk), 32), 21)

for j in range(32):

c[j] = c[j] ^ dk[j]

# 3) step 3

bk = (self.to\_int(b) + self.to\_int(tact\_keys[7 \* (i + 1) - 4 - 1])) % (2 \*\* 32)

bk = self.g(self.add\_zeros(self.to\_list(bk), 32), 13)

diff = self.to\_int(a) - self.to\_int(bk)

if diff < 0:

diff += 2 \*\* 32

a = self.add\_zeros(self.to\_list(diff), 32)

# 4) step 4

e = [0 for \_ in range(32)]

bck = (self.to\_int(b) + self.to\_int(c) + self.to\_int(tact\_keys[7 \* (i + 1) - 3 - 1])) % (2 \*\* 32)

bck = self.g(self.add\_zeros(self.to\_list(bck), 32), 21)

i\_mod = self.add\_zeros(self.to\_list(i + 1), 32)

for j in range(32):

e[j] = bck[j] ^ i\_mod[j]

# 5) step 5

be = (self.to\_int(b) + self.to\_int(e)) % (2 \*\* 32)

b = self.add\_zeros(self.to\_list(be), 32)

# 6) step 6

ce = self.to\_int(c) - self.to\_int(e)

if ce < 0:

ce += 2 \*\* 32

c = self.add\_zeros(self.to\_list(ce), 32)

# 7) step 7

ck = (self.to\_int(c) + self.to\_int(tact\_keys[7 \* (i + 1) - 2 - 1])) % (2 \*\* 32)

ck = self.g(self.add\_zeros(self.to\_list(ck), 32), 13)

d = self.add\_zeros(

self.to\_list((self.to\_int(d) + self.to\_int(ck)) % (2 \*\* 32)),

32

)

# 8) self.add\_zeros(self.to\_list(be), 32)step 8

ak = (self.to\_int(a) + self.to\_int(tact\_keys[7 \* (i + 1) - 1 - 1])) % (2 \*\* 32)

ak = self.g(self.add\_zeros(self.to\_list(ak), 32), 21)

for j in range(32):

b[j] = b[j] ^ ak[j]

# 9) step 9

dk = (self.to\_int(d) + self.to\_int(tact\_keys[7 \* (i + 1) - 1])) % (2 \*\* 32)

dk = self.g(self.add\_zeros(self.to\_list(dk), 32), 5)

for j in range(32):

c[j] = c[j] ^ dk[j]

a, b = b, a

c, d = d, c

b, c = c, b

return b + d + a + c

@staticmethod

def to\_int(lst):

return int("".join(str(\_) for \_ in lst), 2)

@staticmethod

def to\_list(n):

return [int(i) for i in "{0:b}".format(n)]

def decrypt(self, data, key):

if type(data) != list:

raise RuntimeError('Not a list')

if len(data) % 128:

raise RuntimeError('Wrong size')

if type(data) != list:

raise RuntimeError('Not list')

n = len(data)

key = self.add\_zeros(key, 256)

res = []

for i in range(0, n, 128):

res += self.decrypt\_128(data[i:i + 128], key)

while res[0] != 1:

res = res[1:]

return res[1:]

def decrypt\_128(self, data, key):

x = []

for i in range(0, 128, 32):

x.append(data[i:i + 32])

a, b, c, d = x[0], x[1], x[2], x[3]

keys = self.get\_8\_keys(key)

tact\_keys = []

for i in range(7):

tact\_keys += keys

for i in range(7, -1, -1):

# 1) step 1

ak = (self.to\_int(a) + self.to\_int(tact\_keys[7 \* (i + 1) - 1])) % (2 \*\* 32)

ak = self.g(self.add\_zeros(self.to\_list(ak), 32), 5)

for j in range(32):

b[j] = b[j] ^ ak[j]

# 2) step 2

dk = (self.to\_int(d) + self.to\_int(tact\_keys[7 \* (i + 1) - 1 - 1])) % 2 \*\* 32

dk = self.g(self.add\_zeros(self.to\_list(dk), 32), 21)

for j in range(32):

c[j] = c[j] ^ dk[j]

# 3) step 3

bk = (self.to\_int(b) + self.to\_int(tact\_keys[7 \* (i + 1) - 2 - 1])) % (2 \*\* 32)

bk = self.g(self.add\_zeros(self.to\_list(bk), 32), 13)

diff = self.to\_int(a) - self.to\_int(bk)

if diff < 0:

diff += 2 \*\* 32

a = self.add\_zeros(self.to\_list(diff), 32)

# 4) step 4

e = [0 for \_ in range(32)]

bck = (self.to\_int(b) + self.to\_int(c) + self.to\_int(tact\_keys[7 \* (i + 1) - 3 - 1])) % (2 \*\* 32)

bck = self.g(self.add\_zeros(self.to\_list(bck), 32), 21)

i\_mod = self.add\_zeros(self.to\_list(i + 1), 32)

for j in range(32):

e[j] = bck[j] ^ i\_mod[j]

# 5) step 5

be = (self.to\_int(b) + self.to\_int(e)) % (2 \*\* 32)

b = self.add\_zeros(self.to\_list(be), 32)

# 6) step 6

ce = self.to\_int(c) - self.to\_int(e)

if ce < 0:

ce += 2 \*\* 32

c = self.add\_zeros(self.to\_list(ce), 32)

# 7) step 7

ck = (self.to\_int(c) + self.to\_int(tact\_keys[7 \* (i + 1) - 4 - 1])) % (2 \*\* 32)

ck = self.g(self.add\_zeros(self.to\_list(ck), 32), 13)

d = self.add\_zeros(

self.to\_list((self.to\_int(d) + self.to\_int(ck)) % (2 \*\* 32)),

32

)

# 8) step 8

ak = (self.to\_int(a) + self.to\_int(tact\_keys[7 \* (i + 1) - 5 - 1])) % (2 \*\* 32)

ak = self.g(self.add\_zeros(self.to\_list(ak), 32), 21)

for j in range(32):

b[j] = b[j] ^ ak[j]

# 9) step 9

dk = (self.to\_int(d) + self.to\_int(tact\_keys[7 \* (i + 1) - 6 - 1])) % (2 \*\* 32)

dk = self.g(self.add\_zeros(self.to\_list(dk), 32), 5)

for j in range(32):

c[j] = c[j] ^ dk[j]

a, b = b, a

c, d = d, c

a, d = d, a

return c + a + d + b

def g(self, u, r):

if r != 5 and r != 13 and r != 21:

raise RuntimeError('Invalid param r')

if len(u) != 32:

raise RuntimeError('Invalid u len')

x = []

for i in range(0, 32, 8):

x.append(u[i:i+8])

res = []

for u\_i in x:

u\_ir = self.to\_int(u\_i[:4])

u\_ic = self.to\_int(u\_i[4:])

res += self.add\_zeros(

self.to\_list(self.H[u\_ir][u\_ic]),

8

)

if len(res) != 32:

raise RuntimeError('Invalid res len after operations made in G')

return res[r:] + res[:r]

@staticmethod

def reorder\_internal(data):

if len(data) != 32:

raise RuntimeError

res = [0]\*32

res[0:8] = data[24:32]

res[8:16] = data[16:24]

res[16:24] = data[8:16]

res[24:32] = data[0:8]

return res

def reorder(self, data):

if len(data) % 32:

raise RuntimeError('Wrong size in reorder')

for i in range(0, len(data), 32):

data[i:i+32] = self.reorder\_internal(data[i:i+32])

return data

def print\_with\_reorder(self, val):

if len(val) != 32:

raise RuntimeError

print(hex(self.to\_int(self.reorder\_internal(val))))

def encrypt\_in\_chaining\_mode(self, data, key, s):

if type(data) != list:

raise RuntimeError('Not list')

key = self.add\_zeros(key, 256)

data = [1] + data

n = len(data)

m = 0

while m < n:

m += 128

data = self.add\_zeros(data, m)

y0 = self.encrypt\_128(s, key)

y = [y0]

k = 0

for i in range(0, m, 128):

x\_i = data[i:i+128]

xy = [0 for \_ in range(128)]

for j in range(128):

xy[j] = x\_i[j] ^ y[k][j]

y.append(self.encrypt\_128(xy, key))

k += 1

y = y[1:]

res = []

for l in y:

res += l

return res

def decrypt\_from\_chaining\_mode(self, data, key, s):

if type(data) != list:

raise RuntimeError('Not a list')

if len(data) % 128:

raise RuntimeError('Wrong size')

if type(data) != list:

raise RuntimeError('Not list')

n = len(data)

key = self.add\_zeros(key, 256)

x0 = self.encrypt\_128(s, key)

x = [x0]

res = []

k = 0

for i in range(0, n, 128):

dec\_x\_i = self.decrypt\_128(data[i:i + 128], key)

xx = [0 for \_ in range(128)]

for j in range(128):

xx[j] = dec\_x\_i[j] ^ x[k][j]

res += xx

x.append(data[i:i + 128])

k += 1

while res[0] != 1:

res = res[1:]

return res[1:]

def encrypt\_in\_gamma\_mode(self, data, key, s):

if type(data) != list:

raise RuntimeError('Not list')

key = self.add\_zeros(key, 256)

data = [1] + data

n = len(data)

m = 0

while m < n:

m += 128

data = self.add\_zeros(data, m)

y0 = s

y = [y0]

k = 0

for i in range(0, m, 128):

x\_i = data[i:i+128]

ly = self.encrypt(y[k], key)[:128]

xly = [0 for \_ in range(128)]

for j in range(128):

xly[j] = x\_i[j] ^ ly[j]

y.append(xly)

k += 1

y = y[1:]

res = []

for l in y:

res += l

return res

def decrypt\_from\_gamma\_mode(self, data, key, s):

if type(data) != list:

raise RuntimeError('Not a list')

if len(data) % 128:

raise RuntimeError('Wrong size')

if type(data) != list:

raise RuntimeError('Not list')

n = len(data)

key = self.add\_zeros(key, 256)

x0 = s

x = [x0]

res = []

k = 0

for i in range(0, n, 128):

x\_i = data[i:i + 128]

xlx = [0 for \_ in range(128)]

xl = self.encrypt(x[k], key)[:128]

for j in range(128):

xlx[j] = x\_i[j] ^ xl[j]

x.append(x\_i)

res += xlx

k += 1

while res[0] != 1:

res = res[1:]

return res[1:]

def encrypt\_in\_counter\_mode(self, data, key, s):

if type(data) != list:

raise RuntimeError('Not list')

key = self.add\_zeros(key, 256)

data = [1] + data

n = len(data)

m = 0

while m < n:

m += 128

data = self.add\_zeros(data, m)

s = self.encrypt\_128(s, key)

res = []

for i in range(0, n, 128):

x\_i = data[i:i + 128]

s = self.to\_list((self.to\_int(s) + 1) % (2 \*\* 128))

fs = self.encrypt(s, key)[:128]

xlx = [0 for \_ in range(128)]

for j in range(128):

xlx[j] = x\_i[j] ^ fs[j]

res += xlx

return res

def decrypt\_from\_counter\_mode(self, data, key, s):

if type(data) != list:

raise RuntimeError('Not a list')

if len(data) % 128:

raise RuntimeError('Wrong size')

if type(data) != list:

raise RuntimeError('Not list')

n = len(data)

key = self.add\_zeros(key, 256)

s = self.encrypt\_128(s, key)

res = []

for i in range(0, n, 128):

x\_i = data[i:i + 128]

s = self.to\_list((self.to\_int(s) + 1) % (2 \*\* 128))

fs = self.encrypt(s, key)[:128]

xlx = [0 for \_ in range(128)]

for j in range(128):

xlx[j] = x\_i[j] ^ fs[j]

res += xlx

while res[0] != 1:

res = res[1:]

return res[1:]