# Помехоустойчивое шифрование

**(noise\_resistant\_encryption)**

### Шифрование данных в канал

Алгоритмы шифрования и дешифрования основаны на матричной логике со спецификой алгоритмов шифрования. Матрица алгоритма шифрования дана ниже (таблица 1).

Таблица 1 – порождающая матрица расширенного кода Голея

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 11 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 |
| 10 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 |
| 9 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 |
| 8 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| 3 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 2 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Для шифрования данных 12 битного регистра необходимо умножить его как вектор на матрицу. В результате получится вектор длинной 24 бита. Рассмотрим внимательно процесс умножения на матрицу. Первым делом мы сопоставляем вектор, который хотим умножить, с очередным столбцом матрицы. Например, нам нужно зашифровать вектор S = 12’b111000111000.

|  |  |  |
| --- | --- | --- |
| 1 |  | 1 |
| 1 |  | 0 |
| 1 |  | 0 |
| 0 |  | 0 |
| 0 |  | 0 |
| 0 |  | 0 |
| 1 |  | 0 |
| 1 |  | 0 |
| 1 |  | 0 |
| 0 |  | 0 |
| 0 |  | 0 |
| 0 |  | 0 |

Левый – вектор для шифрования Правый – первый столбец матрицы

Далее мы умножаем сопоставленные элементы с помощью операнда &(И). получается столбец:

|  |
| --- |
| 1 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |
| 0 |

После этого мы делаем XOR между всеми элементами полученного вектора. Этот XOR будет равен 1. Это будет значение 23 бита в финальном регистре [23:0].

Можно заметить 2 вещи для облегчения работы.

1 – за каждый полученный бит в финальном векторе отвечает каждый соответствующий по номеру столбец матрицы.

2 – биты в матрице показывают, какие биты в векторе шифрования рабочие. Например, в примере мы видим, что только от первого бита изначального вектора будет зависеть результирующий бит.

## Представлены 2 реализации схем шифрования

1 схема представлена как асинхронная на распределенной памяти. Выдает результат в тот же такт.

2 схема представлена как синхронная на блочно-регистровой памяти.

Выдает результат на следующий такт. При этом используются флаги Enable для подтверждения корректности вводимых в канал данных, ready, показывающий что модуль готов к обработке поступаемых данных, finish, показывающий, что результат на выходе сформирован корректно и данные можно забирать.

### Дешифрование данных из канала

Дешифрование производится несколько сложнее. Помимо умножения на матрицу еще необходимо найти вектор исправления ошибок и умножить на него вектор, пришедший из канала без преображения. Расшифрованными данными будут являться первые 12 бит получившегося регистра.

Теперь разберемся поэтапно:

1. Преобразование вектора из канала путем преобразования вектора из канала путем умножения на матрицу обратного преобразования. Матрица будет представлена в таблице (таблица 2)

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 11 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 10 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 8 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 4 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 2 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Данный этап очень похож на этап шифрования. Поэтому мы не будем его дополнительно обсуждать. Выходной вектор после преобразования будет иметь 12 бит.

1. Вектор, полученный в предыдущем действии, является адресов для таблицы преобразования. В данной таблице будут храниться вектора ошибок, которые необходимо получить по адресу.

Сама таблица была сгенерирована с помощью языка программирования python. Код в приложении 1.

Приложение 1- код генерации таблицы на языке python

|  |
| --- |
| def dec\_to\_bit\_24(chislo):  strok = ""  for i in range(24):  strok += str(chislo % 2)  chislo = chislo // 2  return strok[::-1]  def dec\_to\_bit\_12(chislo):  strok = ""  for i in range(12):  strok += str(chislo % 2)  chislo = chislo // 2  return strok[::-1]  def bit\_to\_dec(chislo):  strok = 0  for i in range(23, -1, -1):  strok += 2 \*\* i \* int(chislo[i])  return strok  massiv = [  #23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0  [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 0, 0,0,0,0,0,0,0,0,0,0],  [1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0,0,0,0,0,0,0,0,0,0],  [1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 0, 0, 1,0,0,0,0,0,0,0,0,0],  [0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 0,1,0,0,0,0,0,0,0,0],  [1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0,0,1,0,0,0,0,0,0,0],  [1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0,0,0,1,0,0,0,0,0,0],  [1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 0, 0, 0,0,0,0,1,0,0,0,0,0],  [0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 0,0,0,0,0,1,0,0,0,0],  [0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0,0,0,0,0,0,1,0,0,0],  [0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0,0,0,0,0,0,0,1,0,0],  [1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0,0,0,0,0,0,0,0,1,0],  [0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0,0,0,0,0,0,0,0,0,1]  ]      massiv\_err = []  massiv\_addres = []  def multiplex(error):  addres = ""  for i in range(len(massiv)):  counter\_one = 0  for j in range(len(massiv[0])):  if((massiv[i][j] == int(error[j])) and massiv[i][j] == 1):  counter\_one += 1  if(counter\_one % 2 == 1):  addres += "1"  else:  addres += "0"  return addres  for i in range(4096 \* 4096):  chis = dec\_to\_bit\_24(i)  if(chis.count("1") <= 4):  addres = multiplex(chis)  if(addres in massiv\_addres):  if(chis.count("1") == 4):  massiv\_addres.append(addres)  massiv\_err.append("0")  else:  massiv\_addres.append(addres)  massiv\_err.append(chis)  new\_err = []  new\_add = []  for i in range(len(massiv\_addres)):  if(massiv\_err[i] != "0"):  new\_add.append(massiv\_addres[i])  new\_err.append(massiv\_err[i])    #for i in range(1800, len(massiv\_err)):  # print(massiv\_addres[i], massiv\_err[i])    def proverka\_na\_blud(massiv):  for i in range(len(massiv) - 1):  addres = massiv[i]  for j in range(i + 1, len(massiv)):  if(addres == massiv[j]):  print("error")  return  carteg = zip(new\_add, new\_err)  sorted\_carteg = sorted(carteg, key=lambda tup: tup[0])  file = open('123', 'w+')  for i in range(4096):    file.write(str(sorted\_carteg[i][1]) + " \n")  file.close()  print("stop") |

1. Вектор ошибок будет умножен на вектор, пришедший из канала. После этого мы выделим 12 первых полученных битов результата. Это будет финальный результат расшифрования.

## Представлены 2 реализации схем шифрования

Обе они довольно похожи. Но первая выполняет действия за такт. Используя где возможно распределенную память. Вторая же почти полностью использует блочную память и выполняет все за 3 такта.