Изучение блочных шифров

Выполнила: Макаров Алексей Игоревич, 3 курс 4 группа 2 подгруппа

2024

**Теоритические сведения**

Блочные шифры (БШ) строятся на основе подстановочных и перестановочных шифров, т. е. являются комбинационными. БШ относятся также к классу симметричных.

Блочное зашифрование (расшифрование) предполагает разбиение исходного открытого (зашифрованного) текста на равные блоки, к которым применяется однотипная процедура зашифрования (расшифрования). Указанная однотипность характеризуется прежде всего тем, что процедура зашифрования (расшифрования) состоит из совокупности повторяющихся наборов преобразований, называемых раундами.

Алгоритм DES

Входной блок данных, состоящий из 64 битов, преобразуется в выходной блок идентичной длины. В алгоритме широко используются рассеивания (подстановки) и перестановки битов текста, о которых мы упоминали выше. Комбинация двух указанных методов преобразования образует фундаментальный строительный блок DES, называемый раундом или циклом. Один блок данных подвергается преобразованию (и при зашифровании, и при расшифровании) в течение 16 раундов. После первоначальной перестановки и разделения 64-битного блока данных на правую (R0) и левую (L0) половины длиной по 32 бита выполняются 16 раундов одинаковых действий.

Слабые и полуслабые ключи. Из-за того что первоначальный ключ изменяется при получении подключа для каждого раунда алгоритма, определенные первоначальные ключи являются слабыми. Вспомним, что первоначальное значение разделяется на две половины, каждая из которых сдвигается независимо. Если все биты каждой половины равны 0 или 1, то для всех раундов алгоритма используется один и тот же ключ. Это может произойти, если ключ состоит из одних 1, из одних 0, или если одна половина ключа состоит из одних 1, а другая – из одних 0. Кроме того, некоторые пары ключей при зашифровании переводят открытый текст в идентичный шифртекст. Иными словами, один из ключей пары может расшифровать сообщения, зашифрованные другим ключом пары.

**Ход работы**

Вариант 8

В данной работе было необходимо разработать авторское приложение в соответствии с целью лабораторной работы. Код программы представлен в листинге 1.

|  |
| --- |
| from Cryptodome.Cipher import DES3  from Cryptodome.Random import get\_random\_bytes  import time  # Функция для разделения данных на блоки  def split\_into\_blocks(data, block\_size):  # Вычисляем количество блоков, которые будут созданы  num\_blocks = len(data) // block\_size  # Определяем, нужно ли добавить дополнение  if len(data) % block\_size != 0:  num\_blocks += 1  # Создаем список для хранения блоков  blocks = []  # Разбиваем данные на блоки  for i in range(num\_blocks):  start = i \* block\_size  end = start + block\_size  block = data[start:end]  # Если это последний блок и данные не делятся нацело, добавляем дополнение  if i == num\_blocks - 1 and len(block) < block\_size:  padding\_length = block\_size - len(block)  padding = bytes([padding\_length]) \* padding\_length  block += padding  blocks.append(block)  return blocks  # Преобразование ключевой информации  def generate\_key(surname):  # Ensure the key length is suitable for Triple DES (either 16 or 24 bytes)  key = surname.encode('utf-8')  if len(key) < 16:  # Pad the key to 16 bytes if it's shorter  key = key.ljust(16, b'\0')  elif len(key) > 24:  # Truncate the key to 24 bytes if it's longer  key = key[:24]  return key  # Шифрование данных  def encrypt(data, key):  cipher = DES3.new(key, DES3.MODE\_ECB)  # Ensure the data is padded to the block size  padded\_data = pad\_data(data, cipher.block\_size)  return cipher.encrypt(padded\_data)  # Расшифрование данных  def decrypt(data, key):  cipher = DES3.new(key, DES3.MODE\_ECB)  return cipher.decrypt(data)  # Замер времени выполнения операции  def measure\_execution\_time(func, \*args):  start\_time = time.time()  result = func(\*args)  end\_time = time.time()  return result, end\_time - start\_time  # Функция для дополнения данных до кратного размера блока  def pad\_data(data, block\_size):  padding\_length = block\_size - (len(data) % block\_size)  padding = bytes([padding\_length]) \* padding\_length  return data + padding  # Функция для анализа лавинного эффекта  def avalanche\_analysis(original\_word, encrypted\_word):  # Реализация анализа лавинного эффекта  pass  # Пример использования  if \_\_name\_\_ == "\_\_main\_\_":  data = b"pivo pit vkusno zhit" # Ваши данные  key = generate\_key("MAKAROV") # Преобразуем вашу фамилию в ключ  encrypted\_data, encryption\_time = measure\_execution\_time(encrypt, data, key) # Шифруем данные  decrypted\_data, decryption\_time = measure\_execution\_time(decrypt, encrypted\_data, key) # Расшифровываем данные  print("Original Data:", data)  print("Encrypted Data:", encrypted\_data)  print("Decrypted Data:", decrypted\_data)  print("Encryption Time:", encryption\_time)  print("Decryption Time:", decryption\_time)  # Пример анализа лавинного эффекта  avalanche\_analysis(data, encrypted\_data) |

Листинг 1 – Программная реализация приложения

Результат работы программы представлен в листинге 2:

|  |
| --- |
| Original Data: b'pivo pit vkusno zhit'  Encrypted Data: b'%,3\xbe\xbft\x98N\x9cB\x19j\x01\x84\xe6\xef\xfa\x8d<J\x0f\x85\x86\xf6'  Decrypted Data: b'pivo pit vkusno zhit\x04\x04\x04\x04'  Encryption Time: 0.0010161399841308594  Decryption Time: 3.886222839355469e-05 |

Листинг 2 – Результат работы приложения

**Вывод**: таким образом, были закреплены знания по алгебраическому описанию, алгоритмам реализации операций зашифрования/расшифрования и оценке криптостойкости блочных шифров.