УДК 004.056.55

РАЗРАБОТКА И РЕАЛИЗАЦИЯ МОДИФИЦИРОВАННОГО АЛГОРИТМА БЛОЧНОГО ШИФРА TEA НА ЯЗЫКЕ PYTHON

Клекта С. А., Чесноков Н. И., Черкесова Л. В, Сафарьян О. А.

Донской государственный технический университет, Ростов-на-Дону, Российская Федерация

seregaklyokta@mail.ru
4esnog96@gmail.com
chia2002@inbox.ru
safari_2006@mail.ru

В рамках данной статьи подробно рассмотрена реализация на языке программирования Python алгоритма шифрования Tiny Encryption Algorithm (TEA), комбинированного с алгоритмом создания раундовых ключей шифра DES. Представлены все возможности программы и интерфейс взаимодействия с ней. Описаны архитектурные решения, показана суть функционального стиля программирования, обоснован выбор стиля для разработки рассматриваемой реализации. Детально описаны модули программы, назначение и алгоритм работы каждого из основных ее структурных элементов.

Ключевые слова: блочный шифр, криптоанализ, ключ шифрования, алгоритм, язык программирования, Python.

UDC 004.056.55

DEVELOPMENT AND IMPLEMENTATION OF THE MODIFIED ALGORITHM OF BLOCK CIPHER TEA IN THE PROGRAMMING LANGUAGE PYTHON

Klyokta S. A., Chesnokov N. I., Cherkesova L. V., Safaryan O. A.,

Don State Technical University, Rostov-on-Don, Russian Federation

seregaklyokta@mail.ru
4esnog96@gmail.com
chia2002@inbox.ru
safari_2006@mail.ru

In this article, the software implementation in Python programming language of the Tiny Encryption Algorithm (TEA) combined with the algorithm for creating the round keys of the DES cipher was considered in detail. All the features of the program and the interface to interact with it presented. The architectural solutions, the essence of the functional programming style, the rationale for choosing this style for writing the implementation under consideration are described. The modules of the program, the purpose and algorithm of operation of each of its main structural elements are described in detail.

Keywords: block cipher, crypto analysis, key of ciphering, algorithm, programming language, Python.

Введение. В течение длительного времени остается актуальным вопрос обеспечения сохранности информации путем ее преобразования, которое исключает возможность доступа посторонних лиц.

Востребованность возможностей криптографических методов в информационных системах обусловлена распространением информационных технологий во всех сферах жизни, вследствие чего пользователям (частным лицам и организациям) приходится хранить конфиденциальную информацию и обмениваться ею. Безопасность в данном случае обеспечивается шифрованием, что подразумевает такое преобразование информации, при котором доступ к ней имеют только легитимные пользователи.

Разработано множество алгоритмов шифрования — как простых, самостоятельных, так и составных [1, 2]. Следует особенно отметить семейство шифров на основе сети Фейстеля — циклической структуры, на каждой итерации которой совершаются однотипные, но не

одинаковые преобразования над ключом шифрования и данными. Такие итерации также называют ячейками или раундами сети Фейстеля.

Основная часть

Из множества шифров на основе сети Фейстеля можно выделить TEA, совмещенный с алгоритмом создания раундовых ключей шифра DES, о специфике работы которого и пойдет речь [1].

Использование программы. Интерфейс командной строки

Рассматриваемая программная реализация шифра представляет собой скрипт-утилиту с интерфейсом командной строки. Синтаксис команды, запускающей алгоритм, имеет следующий вид [3]:

./main.py -k KEY [-h] [-m {encrypt,decrypt}] [-p PATH] [-d DATA] [-t] [--run-tests] [-i]

Строки или символы, начинающиеся с одного или двух дефисов (символ «-»), являются ключами аргументов, принимаемых программой. Часть ключей передается без дополнительных данных, другая часть предполагает указание значения аргумента сразу после ключа, через пробел. В квадратные скобки заключены необязательные аргументы [4].

Пример запуска алгоритма шифрования текста «hello» ключом «1234567»:

./main.py -k 1234567 --data hello

Ниже приведено описание всех возможных аргументов, указываемых при запуске программы. Аргументы могут быть обязательными или предъявлять дополнительные требования к передаваемым значениям. Несоблюдение любого из требований приводит к завершению программы до начала исполнения алгоритма шифрования. При этом в консоль выводится текст ошибки, поясняющий причину преждевременного завершения программы [5].

- -k (--key) KEY (обязательный) — ключ шифрования. KEY — место для ввода ключа шифрования. Минимальная допустимая длина ключа — 7 символов.
- -p (--path) PATH (условно-обязательный) — путь к файлу, содержащему данные, которые необходимо зашифровать или расшифровать.
- -d (--data) DATA (условно-обязательный) — строка, содержащая данные, которые необходимо зашифровать или расшифровать.

Каждый из параметров -d (--data) и -p (--path) является необязательным сам по себе, но наличие хотя бы одного из них обязательно. Если при запуске программы не указаны ни путь к файлу с данными, ни сами данные, программа завершится с ошибкой.

Если указаны и путь к файлу (-p, --path), и данные в виде строки (-d, --data), будут использованы данные из файла, а параметр -d (--data) будет проигнорирован.

- -m (--mode) MODE — режим, в котором будет работать программа (зашифровывать или расшифровывать данные). Возможные значения: encrypt, decrypt; значение по умолчанию: encrypt. При вводе любого значения, не входящего в перечень возможных, программа завершится с ошибкой, указав список допустимых значений.
- -t (--time) — флаг, при указании которого помимо основной функции программа будет замерять время работы в мс, а по окончании выведет его на экран. Аргументы-флаги не принимают никаких значений, в отличие от описанных ранее аргументов.
- -i (--ignore-result) — флаг, при указании которого программа будет работать так же, как и без него, но по окончании проигнорирует результат работы и не выведет его на экран. Данную опцию удобно использовать совместно с флагом -t при измерении производительности программы, когда интересен не результат шифрования, а время работы алгоритма.

- --run-tests — флаг, при указании которого программа не исполняет основную функцию (шифрование или расшифрование), а запускает программные тесты: временной и на наличие эквивалентных ключей.

Архитектура программной реализации алгоритма

Алгоритм шифрования — это почти всегда функция от двух параметров: ключа шифрования и открытого или закрытого текста. Часто это сложный алгоритм, состоящий из более простых алгоритмов, которые можно рассматривать и как функции. Таким образом, можно считать, что основной алгоритм — функция, полученная с помощью композиции более простых функций [4–6].

Становится понятно, что функциональный подход позволяет правильнее и проще программно реализовать алгоритм. Код рассматриваемой реализации модифицированного алгоритма TEA написан именно в функциональном стиле [7].

На рис. 1 показана полная структура программы, отражающая схему композиции функций, составляющих алгоритм.

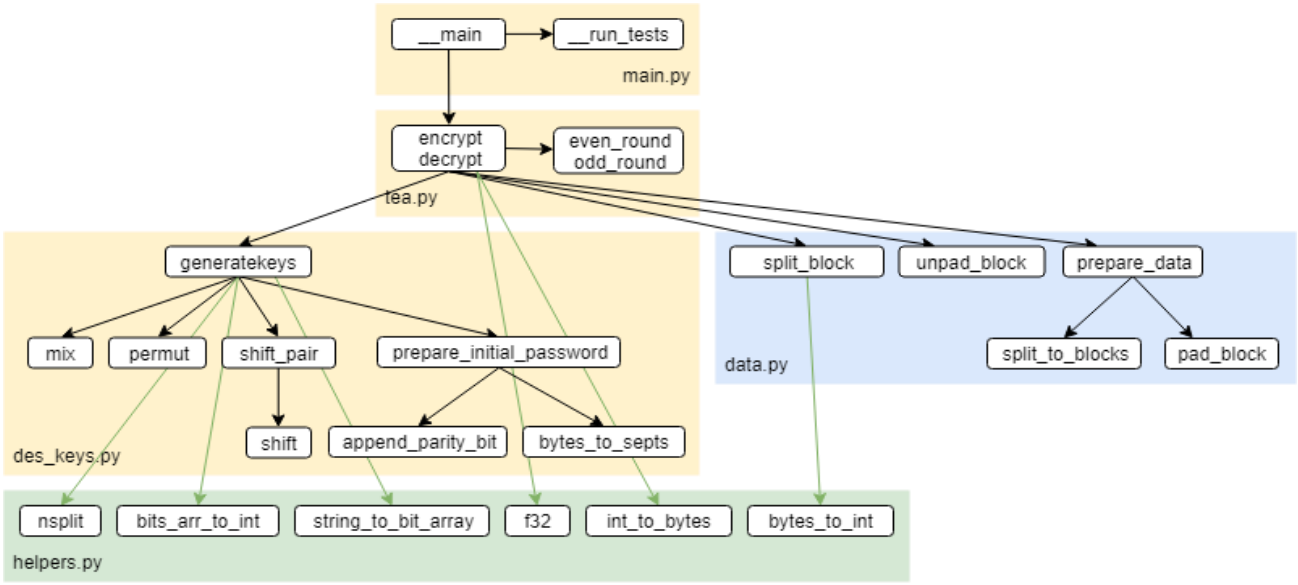

Рис. 1. Структура зависимостей всех функций программной реализации модифицированного алгоритма TEA

Цветными областями обозначены отдельные файлы-модули программы, название которых указано в нижней части каждой области. Белыми прямоугольниками со скругленными углами обозначены функции. Стрелки отражают иерархию зависимостей одних функций от других. Разные цвета стрелок (черный и зеленый) введены только для упрощения визуального восприятия схемы. Соответствующие этим стрелкам связи никак не отличаются.

Подробное рассмотрение программной реализации алгоритма

В данной главе рассматриваются назначение и описание каждой из приведенных на рис. 1 функций. Некоторые функции будут рассмотрены подробно, так как они занимают важное место в алгоритме шифрования. Другие функции будут рассмотрены поверхностно, поскольку они либо не имеют большого значения для алгоритма, либо не требуют подробного описания по причине тривиальности.

Модуль main

Модуль main является входным файлом и содержит две функции: __main и __run_tests. Обе эти функции, по сути, уже были рассмотрены выше. __main вызывается при каждом запуске программы и является входной точкой в программу. При вызове производится обработка и валидация аргументов командной строки, а затем в зависимости от аргументов выбираются режим

и настройки работы программы. В режиме тестирования __main вызывает функцию __run_tests, которая запускает программные тесты. В режиме шифрования и расшифрования используются функции encrypt или decrypt соответственно. Обе эти функции содержатся в модуле tea, описание которого приведено далее.

Модуль tea

В данном модуле — 4 функции: encrypt, decrypt, even_round, odd_round. Две первые используются модулем main и являются входными точками модифицированного алгоритма TEA при шифровании и расшифровании соответственно. Структура этих функций почти идентична, но имеет ряд важных отличий. Функции even_round и odd_round используются функциями encrypt и decrypt и представляют собой операции сети Фейстеля на четных и нечетных раундах.

Код функции encrypt представлен на следующем листинге.

```
def encrypt(data, key):
  keys = generatekeys(key)
  data_blocks = prepare_data(data)
  encrypted = bytearray()

  for block in data_blocks:
    d_sum = 0
    left, right = split_block(block)
    for i in range(CYCLES_COUNT):
      d_sum = f32(d_sum + DELTA)
      left = f32(left + even_round(i, left, right, d_sum, keys))
      right = f32(right + odd_round(i, left, right, d_sum, keys))
    encrypted.extend(int_to_bytes(left, 4))
    encrypted.extend(int_to_bytes(right, 4))

  return bytes(encrypted)
```

Сначала по адаптированному алгоритму создания раундовых ключей DES генерируется набор ключей рассматриваемого алгоритма. Для этого используется функция generatekeys. Далее функцией prepare_data входные данные разбиваются на блоки и инициализируется пустой массив байтов, который будет наполняться шифртекстом.

Функция split_block разделяет каждый блок на «левую» и «правую» половины, после чего начинается цикл раундов сети Фейстеля, в ходе каждой итерации которого половины блоков преобразуются раундовыми функциями even_round и odd_round и суммируются. Все преобразования происходят в поле 2^32, что обеспечивается вспомогательной функцией f32. В каждом цикле используется «магическая» константа DELTA, выведенная из золотого сечения.

Код функции decrypt отличается только направлением цикла и порядком действий на каждой итерации. Далее приведена отличная от функции encrypt часть кода.

```
for i in range(CYCLES_COUNT — 1, -1, -1):
    right = f32(right — odd_round(i, left, right, d_sum, keys))
    left = f32(left — even_round(i, left, right, d_sum, keys))
    d_sum = f32(d_sum — DELTA)
```

Как видно, отличия состоят в том, что номера циклов идут от большего к меньшему, правая и левая части переставлены, а изменение магической константы происходит после преобразований данных. Все это обеспечивает обратное преобразование зашифрованных данных к исходному виду.

Рассмотрим функции раундовых преобразований.

```
def even_round(i: int, left: int, right: int, d_sum: int, keys: List[Tuple[int, int]]) -> int:
  k_even, k_odd = keys[(2 * i)]
  return ((right << 4) + k_even) ^ (right + d_sum) ^ ((right >> 5) + k_odd)

def odd_round(i: int, left: int, right: int, d_sum: int, keys: List[Tuple[int, int]]) -> int:
  k_even, k_odd = keys[(2 * i + 1)]
  return (((left << 4) + k_even) ^ (left + d_sum) ^ ((left >> 5) + k_odd))
```

Обе функции производят одинаковые битовые преобразования, предусмотренные оригинальным алгоритмом TEA, но выбирают разные раундовые ключи и меняют местами левый и правый полублоки.

Модуль data

Модуль data содержит функции prepare_data, split_block, unpad_block, pad_block, split_to_blocks. Все они — вспомогательные, помогают рассмотренным ранее функциям оперировать открытым и зашифрованным текстами.

Функция prepare_data принимает на вход «сырые» данные в виде строки или последовательности байт, введенной пользователем, и возвращает массив блоков по 8 байт, с которыми работает алгоритм. Рассмотрим код функции.

```
def prepare_data(raw_data):
  if type(raw_data) == str:
    data = bytes(raw_data, 'utf-8')
  elif type(raw_data) != bytes and type(raw_data) != bytearray:
    data = bytes(raw_data)
  else:
    data = raw_data

  length = len(data)
  has_semi_block = (length % 8) != 0
  full_blocks_count = length // 8

  data_blocks = split_to_blocks(data)

  if has_semi_block:
    rest_data = pad_block(data[(full_blocks_count * 8):])
    data_blocks.append(rest_data)

  return data_blocks
```

Сначала функция определяет, в каком виде были получены данные, и при необходимости приводит их к последовательности байт. Далее определяется длина входных данных и в зависимости от нее — необходимость дополнить данные «пустыми» байтами до длины, кратной 8 байтам. Последовательность байт функцией split_to_blocks разбивается на блоки, последний блок дополняется «пустыми» байтами, если это необходимо, с помощью функции pad_block.

Функция pad_block принимает на вход неполный блок данных и дополняет его пустыми байтами. Функция unpad_block совершает обратную операцию: обнаруживает в блоке «хвост» из «пустых» байт и обрезает его.

Функция split_to_blocks принимает на вход последовательность байт, определяет количество блоков и возвращает массив блоков по 8 байт. Последний блок может быть неполным, если длина входных данных не кратна 8.

Функция split_block принимает на вход один блок данных длиной 8 байт и возвращает кортеж из двух целых чисел, каждое из которых представляет собой левый или правый полублок в виде битовой последовательности. Преобразовать 4 байта в целое число помогает функция bytes_to_int.

Модуль des_keys

Основная функция модуля des_keys и одна из важнейших частей модифицированного шифра TEA — функция generatekeys.

```
def generatekeys(password: str, cycles=64) -> List[Tuple[int, int]]:
  keys = []
  init_password, parity_bits = prepare_initial_password(password)

  pba = string_to_bit_array([parity_bits])
  pba = permut(pba, PB_EXTENSION)
  pba_l, pba_r = nsplit(pba, 32)

  key = string_to_bit_array(init_password)
  key = permut(key, CP_1)
  key_l, key_r = nsplit(key, 28)

  for i in range(cycles):
    key_l, key_r = shift_pair(key_l, key_r, SHIFT[i])
    pba_l, pba_r = shift_pair(pba_l, pba_r, SHIFT[63 — i])

    tmp_key = key_l + key_r
    tmp_pba = pba_l + pba_r

    pba_start = i % 16
    key_start = (pba_start + 7) % 16

    key_part = permut(tmp_key, CP_2)[key_start:key_start + 32]
    pba_part = permut(tmp_pba, CP_2)[pba_start:pba_start + 32]

    key_pair = mix(key_part, pba_part, (i // 32) + 1)
    keys.append((bits_arr_to_int(key_pair[:32]), bits_arr_to_int(key_pair[32:64])))
  return keys
```

Эта функция принимает на вход ключ в виде строки и возвращает полный набор уникальных раундовых ключей, используемых функциями encrypt и decrypt. Сначала функция prepare_initial_password преобразует 7-символьный строковый пароль к кортежу из преобразованного 8-байтного пароля и набора бит четности, за счет которых пароль был расширен до 8 байт. Далее с помощью вспомогательной функции permut производится перестановка бит пароля по таблице перестановки, а затем пароль разделяется на две половины по 28 бит. Наконец, согласно алгоритму создания раундовых ключей рассматриваемой модификации TEA, создается набор ключей. На каждой итерации выработки раундовых ключей используются функции permut, shift_pair и mix.

Функция permut принимает на вход массив каких-либо элементов и таблицу перестановки, применяет перестановку к массиву элементов и возвращает результат перестановки.

Функция shift_pair применяет к двум переданным массивам и числу-сдвигу функцию shift, которая производит циклический сдвиг элементов массива, переданного первым параметром, на число позиций, переданное вторым параметром.

Функция mix, как и shift_pair, принимает на вход две последовательности и число-период, и объединяет последовательности в новую, длина которой равна сумме длин исходных последовательностей. При объединении новая последовательность формируется так, что элементы исходных чередуются друг с другом по несколько штук, число которых равно последнему параметру.

Функция prepare_initial_password, как упомянуто выше, принимает 7-символьный строковый пароль и возвращает кортеж из преобразованного 8-байтного пароля и набора бит четности.

```
def prepare_initial_password(password: str) -> Tuple[bytes, int]:
  if type(password) == str:
    password = password.encode('utf-8')
  septs = bytes_to_septs(password[:7])
  res = []
  parity_bits = 0
  for sept in septs:
    curr_byte, new_bit = append_parity_bit(sept)
    res.append(curr_byte)
    parity_bits = (parity_bits << 1) | new_bit

  return bytes(res).ljust(8, b'\x00'), parity_bits
```

Прежде всего, принятый строковый пароль преобразуется функцией bytes_to_septs к байтовой последовательности длиной 7 байт. Эта последовательность преобразуется в массив 7-битовых последовательностей, каждая из которых затем с помощью функции append_parity_bit дополняется битом четности. Полученные байты объединяются в преобразованный пароль, а биты четности сохраняются в отдельном байте.

Функция append_parity_bit принимает битовую последовательность в виде целого числа первым аргументом и размер этой последовательности — вторым, необязательным аргументом, при отсутствии которого длина последовательности считается равной 7 битам. Далее рассчитывается число единиц в последовательности. Если число единиц четное, в конец добавляется бит четности, равный 1. Если же единиц нечетное количество, последним битом последовательности становится 0.

Функция возвращает кортеж из двух чисел: первое — 8-битовая последовательность, сформированная добавлением бита четности в конец исходной последовательности; второе — бит четности.

Заключение. В результате проделанной работы удалось создать модификацию алгоритма блочного шифра TEA, которая оказалась более криптостойкой, чем оригинальный TEA.

Библиографический список

1. Разработка комбинированного алгоритма шифрования на примере объединения блочных шифров TEA и DES [Электронный ресурс] / Н. И. Чесноков [и др.] // Современные наукоемкие технологии. — 2018. — № 12, ч. 2. — С. 276–281. — Режим доступа: http://www.top-technologies.ru/ru/article/view?id=37333 (дата обращения: 01.02.19).

2. Панасенко, С. П. Алгоритмы шифрования. Специальный справочник / С. П. Панасенко. — Санкт-Петербург : БХВ-Петербург, 2009. — 576 с.

3. Kelsey, J. Key-schedule cryptanalysis of IDEA, G-DES, GOST, SAFER, and Triple-DES / J. Kelsey, B. Schneier, D. Wagner // Lecture Notes in Computer Science. — 1996 — Vol. 1109. — P. 237–251. DOI:10.1007/3-540-68697-5_19.

4. Бабаш, А. В. Криптография / А. В. Бабаш, Г. П. Шанкин ; под ред. В. П. Шерстюка, Э. Л. Применко. — Москва : Солон-Пресс, 2007. — 512 с. — (Аспекты защиты).

5. Kelsey, J. Related-key cryptanalysis of 3-WAY, Biham-DES, CAST, DES-X NewDES, RC2, and TEA / J. Kelsey, B. Schneier, D. Wagner // Lecture Notes in Computer Science. — 1997. — Vol. 1334. — P. 233–246. DOI:10.1007/BFb0028479.

6. Biham, E. New types of cryptanalytic attacks using related keys / E. Biham // Journal of Cryptology. — 1994. — Vol. 7, is. 4. — P. 229–246. DOI:10.1007/BF00203965.

7. Impossible differential cryptanalysis of reduced round XTEA and TEA / D. Moon [et al.] / Lecture Notes in Computer Science. — 2002. — Vol. 2365. — P. 49–60. DOI:10.1007/3-540-45661-9_4.