Московский авиационный институт
(Национальный исследовательский университет)

Лабораторная работа № 3

по курсу «Криптография»

Студент: Фирфаров А. С.
Группа: 8О-308Б

Москва, 2020

Постановка задачи

№0. Строку, в которой записано своё ФИО подать на вход в хеш-функцию ГОСТ Р 34.11-2012 (Стрибог). Младшие 4 бита выхода интерпретировать как число, которое в дальнейшем будет номером варианта. Процесс выбора варианта требуется отразить в отчёте.
№1. Программно реализовать один из алгоритмов функции хеширования в соответствии с номером варианта. Алгоритм содержит в себе несколько раундов.
№2. Модифицировать оригинальный алгоритм таким образом, чтобы количество раундов было настраиваемым параметром программы. в этом случае новый алгоритм не будет являться стандартом, но будет интересен для исследования.
№3. Применить подходы дифференциального криптоанализа к полученным алгоритмам с разным числом раундов.
№4. Построить график зависимости количества раундов и возможности различения отдельных бит при количестве раундов 1,2,3,4,5,... .
№5. Сделать выводы

Примечание №1. Допустимо использовать сторонние реализации для пункта 1, при условии, что они проходят тесты из стандарта и пригодны для дальнейшей модификации.

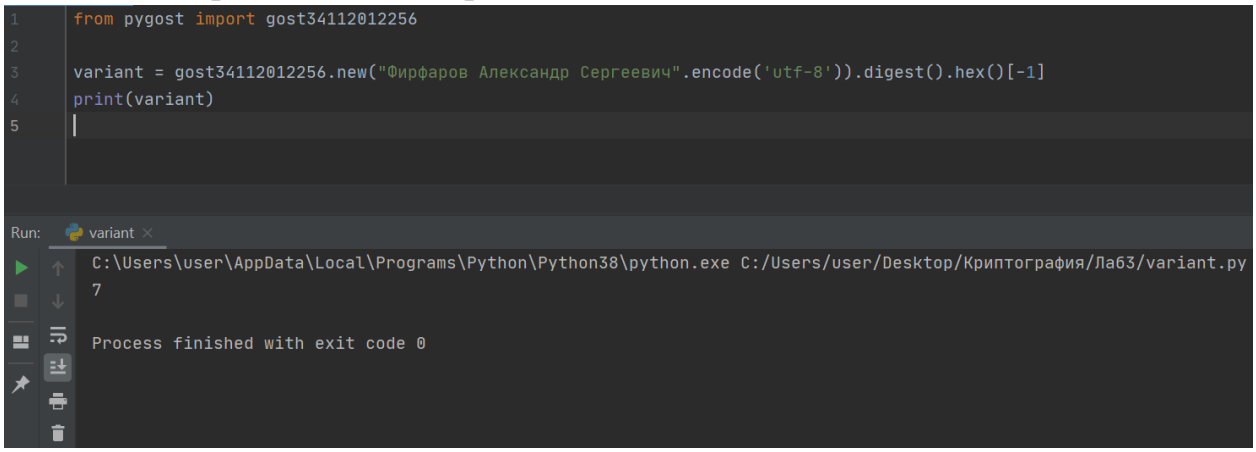
Примечание №2. Если в алгоритме описывается семейство с разными размерами блоков, то можно выбрать любой из них.

Номер варианта == Алгоритм
0 == ГОСТ Р 34.11-94
1 == ГОСТ Р 34.11-2012 (Стрибог)
2 == Luffa
3 == BLAKE
4 == SHA-0
5 == SHA-1
6 == SHA-2
7 == Keccak
8 == JH
9 == Shabal
A == Skein
B == MD4
C == CubeHash

D == MD5
E == SIMD
F == Whirlpool

Метод решения

Сначала я определил свой вариант:



Код на Python:

```
from pygost import gost34112012256

variant = gost34112012256.new("Фирфаров Александр Сергеевич".encode('utf-
8')).digest().hex()[-1]
print(variant)
```

В соответствии с вариантом мне достался алгоритм Keccak. Алгоритм состоит из двух этапов. Первый этап называется Absorbing (впитывание). На данном этапе исходное сообщение подвергается многораундовым перестановкам f. Второй этап называется Squeezing (отжатие). Здесь происходит вывод получившегося в результате перестановок значения Z.

Реализацию алгоритма я взял из github репозитория: https://github.com/mgoffin/keccak-python. Код из данного репозитория содержит реализацию Keccak с возможностью использовать различные функции перестановки (f-25, f-50, f-100, f-200, f-400, f-800, f-1600). По умолчанию используется f-1600. Данная функция имеет наибольшее число раундов, равное 24.

Для настройки количества раундов я модифицировал исходный код. В функцию Keccak я добавил дополнительный параметр num_of_rounds, отвечающий за число раундов:

```
def Keccak(self, M, r=1024, c=576, n=1024, num_of_rounds=None, verbose=False):
"""Compute the Keccak[r,c,d] sponge function on message M

M: message pair (length in bits, string of hex characters ('9AFC...')
r: bitrate in bits (defautl: 1024)
c: capacity in bits (default: 576)
n: length of output in bits (default: 1024),
verbose: print the details of computations(default:False)
"""

# Check the inputs
if (r < 0) or (r % 8 != 0):
    raise KeccakError.KeccakError('r must be a multiple of 8 in this implementation')
if (n % 8 != 0):
    raise KeccakError.KeccakError('outputLength must be a multiple of 8')
self.setB(r + c, num_of_rounds)

. . .
```

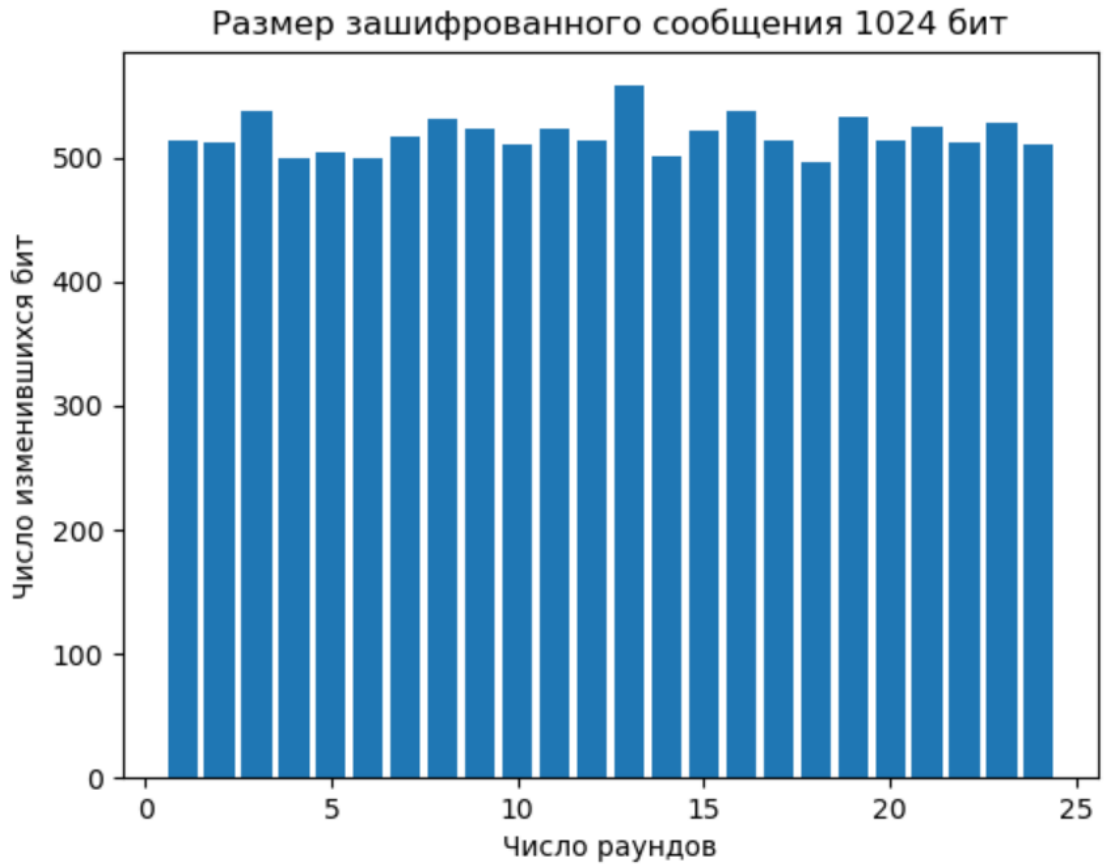
setB устанавливает параметры алгоритма, в том числе количество раундов:

```
def setB(self, b, num_of_rounds=None):
    """Set the value of the parameter b (and thus w,l and nr)

    b: parameter b, must be choosen among [25, 50, 100, 200, 400, 800, 1600]
    """

    if b not in [25, 50, 100, 200, 400, 800, 1600]:
        raise KeccakError.KeccakError('b value not supported - use 25, 50, 100, 200,
400, 800 or 1600')

    # Update all the parameters based on the used value of b
    self.b = b
    self.w = b // 25
    self.l = int(math.log(self.w, 2))

    if num_of_rounds is None:
        self.nr = 12 + 2 * self.l
    else:
        self.nr = num_of_rounds
```

По умолчанию число раундов вычисляется по формуле $nr = 12 + 2 * l$, где $2^l = b/25$.

В программе я формирую одно исходное сообщение, а также другое сообщение, отличающееся от первого 1 случайным битом. Размер сообщений составляет 512 байт. Далее я перевожу оба сообщения в формат hex строки. После этого, для различных значений количества раундов я получаю хеши обоих сообщений и считаю число различающихся битов. В конце программа строит график зависимости количества отличающихся бит от числа раундов.

Результаты работы программы



Код

3.py

```python
import secrets
import bitarray
import matplotlib.pyplot as plt

import Keccak

MAX_NUM_OF_ROUNDS = 24
SIZE_OF_MESSAGE_BYTE = 512
SIZE_OF_ENCRYPTED_MESSAGE_BIT = 1024


def calculate_difference(msg_1, msg_2):
    count = 0

    bytes_1 = bytes.fromhex(msg_1)
    bytes_2 = bytes.fromhex(msg_2)

    ba_1 = bitarray.bitarray()
    ba_1.frombytes(bytes_1)

    ba_2 = bitarray.bitarray()
    ba_2.frombytes(bytes_2)

    for i in range(len(ba_1)):
        if ba_1[i] != ba_2[i]:
            count += 1
    return count


def change_one_bit(message):
    ba = bitarray.bitarray()
    ba.frombytes(message)
    bite_idx = secrets.randbelow(len(ba))
    ba[bite_idx] = True if ba[bite_idx] is False else False
    return ba.tobytes()


def get_random_message(length):
    return secrets.token_bytes(length)


def graph(differences):
    plt.bar([i for i in range(1, MAX_NUM_OF_ROUNDS + 1)], differences)
    plt.title(f'Размер зашифрованного сообщения {SIZE_OF_ENCRYPTED_MESSAGE_BIT} бит')
    plt.xlabel('Число раундов')
    plt.ylabel('Число изменившихся бит')
    plt.show()


def main():
    my_keccak = Keccak.Keccak()

    message_1 = get_random_message(SIZE_OF_MESSAGE_BYTE)
    message_2 = change_one_bit(message_1)
```

```
    message_1 = message_1.hex().upper()
    message_2 = message_2.hex().upper()

    assert len(message_1) == len(message_2)

    differences = []
    for num_round in range(1, MAX_NUM_OF_ROUNDS + 1):
        encrypted_message_1 = my_keccak.Keccak((len(message_1) * 4, message_1),
                                               num_of_rounds=num_round,
n=SIZE_OF_ENCRYPTED_MESSAGE_BIT)
        encrypted_message_2 = my_keccak.Keccak((len(message_1) * 4, message_2),
                                               num_of_rounds=num_round,
n=SIZE_OF_ENCRYPTED_MESSAGE_BIT)
        differences.append(calculate_difference(encrypted_message_1,
encrypted_message_2))

    graph(differences)


if __name__ == '__main__':
    main()
```

Keccak.py

```
import math


class KeccakError(Exception):
    """Class of error used in the Keccak implementation

    Use: raise KeccakError.KeccakError("Text to be displayed")"""

    def __init__(self, value):
        self.value = value

    def __str__(self):
        return repr(self.value)


class Keccak:
    """
    Class implementing the Keccak sponge function
    """

    def __init__(self, b=1600):
        """Constructor:

        b: parameter b, must be 25, 50, 100, 200, 400, 800 or 1600 (default value)"""
        self.setB(b)

    def setB(self, b, num_of_rounds=None):
        """Set the value of the parameter b (and thus w,l and nr)

        b: parameter b, must be choosen among [25, 50, 100, 200, 400, 800, 1600]
        """

        if b not in [25, 50, 100, 200, 400, 800, 1600]:
            raise KeccakError.KeccakError('b value not supported - use 25, 50, 100,
200, 400, 800 or 1600')

        # Update all the parameters based on the used value of b
        self.b = b
```

```
        self.w = b // 25
        self.l = int(math.log(self.w, 2))

        if num_of_rounds is None:
            self.nr = 12 + 2 * self.l
        else:
            self.nr = num_of_rounds

    # Constants

    ## Round constants
    RC = [0x0000000000000001,
          0x0000000000008082,
          0x800000000000808A,
          0x8000000080008000,
          0x000000000000808B,
          0x0000000080000001,
          0x8000000080008081,
          0x8000000000008009,
          0x000000000000008A,
          0x0000000000000088,
          0x0000000080008009,
          0x000000008000000A,
          0x000000008000808B,
          0x800000000000008B,
          0x8000000000008089,
          0x8000000000008003,
          0x8000000000008002,
          0x8000000000000080,
          0x000000000000800A,
          0x800000008000000A,
          0x8000000080008081,
          0x8000000000008080,
          0x0000000080000001,
          0x8000000080008008]

    ## Rotation offsets
    r = [[0, 36, 3, 41, 18],
         [1, 44, 10, 45, 2],
         [62, 6, 43, 15, 61],
         [28, 55, 25, 21, 56],
         [27, 20, 39, 8, 14]]

    ## Generic utility functions

    def rot(self, x, n):
        """Bitwise rotation (to the left) of n bits considering the \
        string of bits is w bits long"""

        n = n % self.w
        return ((x >> (self.w - n)) + (x << n)) % (1 << self.w)

    def fromHexStringToLane(self, string):
        """Convert a string of bytes written in hexadecimal to a lane value"""

        # Check that the string has an even number of characters i.e. whole number of
bytes
        if len(string) % 2 != 0:
            raise KeccakError.KeccakError("The provided string does not end with a
full byte")

        # Perform the modification
```

```
        temp = ''
        nrBytes = len(string) // 2
        for i in range(nrBytes):
            offset = (nrBytes - i - 1) * 2
            temp += string[offset:offset + 2]
        return int(temp, 16)

    def fromLaneToHexString(self, lane):
        """Convert a lane value to a string of bytes written in hexadecimal"""

        laneHexBE = (("%%0%dX" % (self.w // 4)) % lane)
        # Perform the modification
        temp = ''
        nrBytes = len(laneHexBE) // 2
        for i in range(nrBytes):
            offset = (nrBytes - i - 1) * 2
            temp += laneHexBE[offset:offset + 2]
        return temp.upper()

    def printState(self, state, info):
        """Print on screen the state of the sponge function preceded by \
        string info

        state: state of the sponge function
        info: a string of characters used as identifier"""

        print("Current value of state: %s" % (info))
        for y in range(5):
            line = []
            for x in range(5):
                line.append(hex(state[x][y]))
            print('\t%s' % line)

    ### Conversion functions String <-> Table (and vice-versa)

    def convertStrToTable(self, string):
        """Convert a string of bytes to its 5×5 matrix representation

        string: string of bytes of hex-coded bytes (e.g. '9A2C...')"""

        # Check that input paramaters
        if self.w % 8 != 0:
            raise KeccakError("w is not a multiple of 8")
        if len(string) != 2 * (self.b) // 8:
            raise KeccakError.KeccakError("string can't be divided in 25 blocks of w
bits\
            i.e. string must have exactly b bits")

        # Convert
        output = [[0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0],
                  [0, 0, 0, 0, 0]]
        for x in range(5):
            for y in range(5):
                offset = 2 * ((5 * y + x) * self.w) // 8
                output[x][y] = self.fromHexStringToLane(string[offset:offset + (2 *
self.w // 8)])
        return output

    def convertTableToStr(self, table):
```

```
        """Convert a 5×5 matrix representation to its string representation"""

        # Check input format
        if self.w % 8 != 0:
            raise KeccakError.KeccakError("w is not a multiple of 8")
        if (len(table) != 5) or (False in [len(row) == 5 for row in table]):
            raise KeccakError.KeccakError("table must be 5×5")

        # Convert
        output = [''] * 25
        for x in range(5):
            for y in range(5):
                output[5 * y + x] = self.fromLaneToHexString(table[x][y])
        output = ''.join(output).upper()
        return output

    def Round(self, A, RCfixed):
        """Perform one round of computation as defined in the Keccak-f permutation

        A: current state (5×5 matrix)
        RCfixed: value of round constant to use (integer)
        """

        # Initialisation of temporary variables
        B = [[0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0]]
        C = [0, 0, 0, 0, 0]
        D = [0, 0, 0, 0, 0]

        # Theta step
        for x in range(5):
            C[x] = A[x][0] ^ A[x][1] ^ A[x][2] ^ A[x][3] ^ A[x][4]

        for x in range(5):
            D[x] = C[(x - 1) % 5] ^ self.rot(C[(x + 1) % 5], 1)

        for x in range(5):
            for y in range(5):
                A[x][y] = A[x][y] ^ D[x]

        # Rho and Pi steps
        for x in range(5):
            for y in range(5):
                B[y][(2 * x + 3 * y) % 5] = self.rot(A[x][y], self.r[x][y])

        # Chi step
        for x in range(5):
            for y in range(5):
                A[x][y] = B[x][y] ^ ((~B[(x + 1) % 5][y]) & B[(x + 2) % 5][y])

        # Iota step
        A[0][0] = A[0][0] ^ RCfixed

        return A

    def KeccakF(self, A, verbose=False):
        """Perform Keccak-f function on the state A

        A: 5×5 matrix containing the state
```

```
        verbose: a boolean flag activating the printing of intermediate computations
        """

        if verbose:
            self.printState(A, "Before first round")

        for i in range(self.nr):
            # NB: result is truncated to lane size
            A = self.Round(A, self.RC[i] % (1 << self.w))

            if verbose:
                self.printState(A, "Satus end of round #%d/%d" % (i + 1, self.nr))

        return A

    ### Padding rule

    def pad10star1(self, M, n):
        """Pad M with the pad10*1 padding rule to reach a length multiple of r bits

        M: message pair (length in bits, string of hex characters ('9AFC...')
        n: length in bits (must be a multiple of 8)
        Example: pad10star1([60, 'BA594E0FB9EBBD30'],8) returns 'BA594E0FB9EBBD93'
        """

        [my_string_length, my_string] = M

        # Check the parameter n
        if n % 8 != 0:
            raise KeccakError.KeccakError("n must be a multiple of 8")

        # Check the length of the provided string
        if len(my_string) % 2 != 0:
            # Pad with one '0' to reach correct length (don't know test
            # vectors coding)
            my_string = my_string + '0'
        if my_string_length > (len(my_string) // 2 * 8):
            raise KeccakError.KeccakError("the string is too short to contain the
number of bits announced")

        nr_bytes_filled = my_string_length // 8
        nbr_bits_filled = my_string_length % 8
        l = my_string_length % n
        if ((n - 8) <= l <= (n - 2)):
            if (nbr_bits_filled == 0):
                my_byte = 0
            else:
                my_byte = int(my_string[nr_bytes_filled * 2:nr_bytes_filled * 2 + 2],
16)
            my_byte = (my_byte >> (8 - nbr_bits_filled))
            my_byte = my_byte + 2 ** (nbr_bits_filled) + 2 ** 7
            my_byte = "%02X" % my_byte
            my_string = my_string[0:nr_bytes_filled * 2] + my_byte
        else:
            if (nbr_bits_filled == 0):
                my_byte = 0
            else:
                my_byte = int(my_string[nr_bytes_filled * 2:nr_bytes_filled * 2 + 2],
16)
            my_byte = (my_byte >> (8 - nbr_bits_filled))
            my_byte = my_byte + 2 ** (nbr_bits_filled)
            my_byte = "%02X" % my_byte
```

```
            my_string = my_string[0:nr_bytes_filled * 2] + my_byte
            while ((8 * len(my_string) // 2) % n < (n - 8)):
                my_string = my_string + '00'
            my_string = my_string + '80'

        return my_string

    def Keccak(self, M, r=1024, c=576, n=1024, num_of_rounds=None, verbose=False):
        """Compute the Keccak[r,c,d] sponge function on message M

        M: message pair (length in bits, string of hex characters ('9AFC...')
        r: bitrate in bits (defautl: 1024)
        c: capacity in bits (default: 576)
        n: length of output in bits (default: 1024),
        verbose: print the details of computations(default:False)
        """

        # Check the inputs
        if (r < 0) or (r % 8 != 0):
            raise KeccakError.KeccakError('r must be a multiple of 8 in this
implementation')
        if (n % 8 != 0):
            raise KeccakError.KeccakError('outputLength must be a multiple of 8')
        self.setB(r + c, num_of_rounds)

        if verbose:
            print("Create a Keccak function with (r=%d, c=%d (i.e. w=%d))" % (r, c,
(r + c) // 25))

        # Compute lane length (in bits)
        w = (r + c) // 25

        # Initialisation of state
        S = [[0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0],
             [0, 0, 0, 0, 0]]

        # Padding of messages
        P = self.pad10star1(M, r)

        if verbose:
            print("String ready to be absorbed: %s (will be completed by %d x '00')"
% (P, c // 8))

        # Absorbing phase
        for i in range((len(P) * 8 // 2) // r):
            Pi = self.convertStrToTable(P[i * (2 * r // 8):(i + 1) * (2 * r // 8)] +
'00' * (c // 8))

            for y in range(5):
                for x in range(5):
                    S[x][y] = S[x][y] ^ Pi[x][y]
            S = self.KeccakF(S, verbose)

        if verbose:
            print("Value after absorption : %s" % (self.convertTableToStr(S)))

        # Squeezing phase
        Z = ''
        outputLength = n
```

```
        while outputLength > 0:
            string = self.convertTableToStr(S)
            Z = Z + string[:r * 2 // 8]
            outputLength -= r
            if outputLength > 0:
                S = self.KeccakF(S, verbose)

            # NB: done by block of length r, could have to be cut if outputLength
            #     is not a multiple of r

        if verbose:
            print("Value after squeezing : %s" % (self.convertTableToStr(S)))

        return Z[:2 * n // 8]
```

Выводы

В данной лабораторной работе я познакомился с алгоритмом хеширования Keccak. Данный алгоритм стал победителем конкурса криптографических алгоритмов в 2012 году, поэтому можно считать этот алгоритм достаточно эффективным. Из полученных графиков можно заметить, что при изменении исходного сообщения всего лишь на 1 бит, в выходном сообщении меняется примерно половина всех битов. Это выполняется для всех размеров входного и выходного сообщений и не зависит от числа раундов. Можно говорить о том, что данный алгоритм удовлетворяет лавинному критерию, что положительно сказывается на его устойчивости к взлому. Еще одним достоинством алгоритма является то, что его легко модифицировать. Кроме того, он имеет множество параметров для настройки.