## МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

## Федеральное государственное бюджетное образовательное учреждение высшего образования

## «Московский Авиационный Институт» (Национальный Исследовательский Университет)

## Институт: №8 «Информационные технологии и прикладная математика»

## Кафедра: 806 «Вычислительная математика и программирование»

### Курсовой проект

### по курсу «Криптография»

###### Группа: М8О-310Б-21

###### Студент: А. С. Личковаха Преподаватель: А. В. Борисов Оценка:

###### Дата: 17.06.2024

Москва, 2024

## ОГЛАВЛЕНИЕ

1. [Тема 3](#_TOC_250005)
2. [Задание 3](#_TOC_250004)
3. [Теория 4](#_TOC_250003)
4. [Ход лабораторной работы 6](#_TOC_250002)
5. [Выводы 8](#_TOC_250001)
6. [Листинг программы 16](#_TOC_250000)

# Тема

###### Аутентификация с асимметричными алгоритмами шифрования

# Задание

###### Строку в которой записано своё ФИО подать на вход в хеш-функцию ГОСТ Р 34.11-2012 (Стрибог). Младшие 4 бита выхода интерпретировать как 16-тиричное число, которое в дальнейшем будет номером варианта.

###### Программно реализовать один из алгоритмов функции хеширования в соответствии с номером варианта. Алгоритм содержит в себе несколько раундов.

###### 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.

###### Применить подходы дифференциального криптоанализа к полученным алгоритмам с разным числом раундов.

###### Построить график зависимости количества раундов и возможности различения отдельных бит при количестве раундов 1,2,3,4,5,...

###### Сделать выводы.

# Теория

**Концепции построения хэш-функции «Стрибог»**

В соответствии с требованиями, высказанными на конференции РусКрипто-2010, в работе, посвящённой новой хеш-функции: • у новой хеш-функции не должно быть свойств, которые позволяли бы применить известные атаки; • в хеш-функции должны использоваться изученные конструкции и преобразования; • вычисление хеш-функции должно быть эффективным, занимать мало времени; • не должно быть лишних преобразований, усложняющих конструкцию хеш-функции. Причем каждое используемое в хеш-функции преобразование должно отвечать за определённые криптографические свойства. В той же работе вводятся «универсальные» требования, касающиеся трудоемкости атак на хеш-функцию:

* Сложность:
* построение прообраза 2n
* построение коллизии 2n/2
* построение второго прообраза 2n/(длина сообщения)
* удлинение прообраза 2n

**Сравнение ГОСТ Р 34.11-2012 и ГОСТ Р 34.11-94**

• В ГОСТ Р 34.11-2012 размер блоков сообщения и внутреннего состояния хеш-функции составляет 512 бит против 256 бит в ГОСТ Р 34.11-94.

• Новый стандарт определяет две функции хеширования с длинами хеш-кода 256 и 512 бит, в то время как в старом стандарте длина хеш-кода может быть только 256 бит. Возможность вариации выходного хеша может быть полезна в случае встроенных реализаций с ограниченными ресурсами или наличия каких-то дополнительных требований в области криптографии.

• Основное отличие современной хеш-функции от старой — функция сжатия. В ГОСТ Р 34.11-2012 используется функции сжатия, в основе которой лежат три преобразования: нелинейное биективное преобразование (обозначается S), перестановка байт (обозначается P), линейное преобразование (обозначается L). В ГОСТ Р 34.11-94 используется функция сжатия, основанная на симметричном блочном шифре ГОСТ Р 28147-89, также эта функция использует операции перемешивания.

• При вычислении новой хеш-функции, если размер сообщения не кратен размеру обрабатываемого блока (для современного стандарта — 512 бит, для старого стандарта — 256 бит), то такой блок дополняется вектором (00 … 01). При вычислении старой хеш-функции неполный блок дополняется значением (00 … 0). Считается, что дополнение (00 … 01) лучше, чем (00 … 0), с криптографической точки зрения, так как дополнения значением (00 … 0) приводит к атакам Оракула дополнения. В случае ненулевого дополнения была доказана стойкость к подобным атакам.

• Ещё одно отличие состоит в том, что стандарт ГОСТ Р 34.11-94 не определял значение инициализационного вектора, в то время как в стандарте ГОСТ Р 34.11-2012 значение инициализационного вектора фиксировано и определено в стандарте: для хеш-функции с размером выходного хеша 512 бит это вектор (00 … 0), для хеш-функции с размером выходного хеш-кода 256 бит — (000000010 … 100000001) (все байты равны 1).

**Функция сжатия**

В хеш-функции важным элементом является функция сжатия. В ГОСТ Р 34.11-2012 функция сжатия основана на конструкции Миагути — Пренеля. Схема конструкции Миагути — Пренеля: h, m — вектора, поступающие на вход функции сжатия; g(h, m) — результат функции сжатия; E — блочный шифр с длиной блока и ключа 512 бит. В качестве блочного шифра в хеш-функции ГОСТ Р 34.11-2012 взят XSPL-шифр. Этот шифр состоит из следующих преобразований: • сложение по модулю 2; • преобразование замены или подстановки. Обозначается S-преобразование; • преобразование перестановки. Обозначается P-преобразование; • линейное преобразование. Обозначается L-преобразование. Преобразования, используемые в новой хеш-функции, должны быть хорошо изучены. Поэтому в блочном шифре E используются преобразования X, S, P, L, которые хорошо изучены. Важным параметром блочного шифра является то, как выбирается ключ, который будет использовать на каждом раунде. В блочном шифре, используемом в ГОСТ Р 34.11-2012, ключи , , … ,  для каждого из 13 раундов генерируются с помощью самой функции шифрования. , , … ,  — итерационные константы, которые являются 512 битовыми векторам. Их значения указаны в соответствующем разделе стандарта. Описание [править | править код] В основу хеш-функции положена итерационная конструкция Меркла — Дамгора с использованием MD-усиления. Под MD-усилением понимается дополнение неполного блока при вычислении хеш-функции до полного путём добавления вектора (0 … 01) такой длины, чтобы получился полный блок. Из дополнительных элементов нужно отметить следующие: • завершающее преобразование, которое заключается в том, что функция сжатия применяется к контрольной сумме всех блоков сообщения по модулю 2512; • при вычислении хеш-кода на каждой итерации применяются разные функции сжатия. Можно сказать, что функция сжатия зависит от номера итерации. Описанные выше решения позволяют противостоять многим известным атакам. Кратко описание хеш-функции ГОСТ Р 34.11-2012 можно представить следующим образом. На вход хеш-функции подается сообщение произвольного размера. Далее сообщение разбивается на блоки по 512 бит, если размер сообщения не кратен 512, то оно дополняется необходимым количеством бит. Потом итерационно используется функция сжатия, в результате действия которой обновляется внутреннее состояние хеш-функции. Также вычисляется контрольная сумма блоков и число обработанных бит. Когда обработаны все блоки исходного сообщения, производятся ещё два вычисления, которые завершают вычисление хеш-функции: • обработка функцией сжатия блока с общей длиной сообщения. • обработка функцией сжатия блока с контрольной суммой.

# Ход лабораторной работы

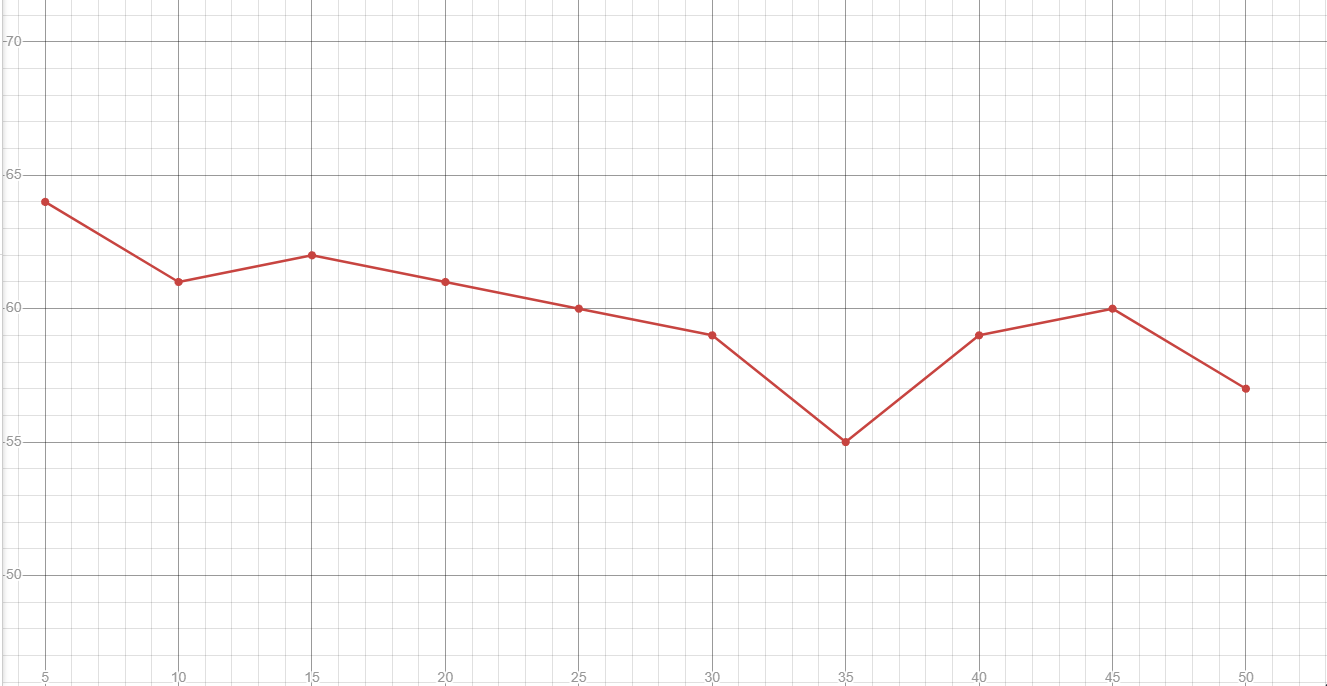
###### **Вариант 1 - ГОСТ Р 34.11-2012**

###### Я реализовал алгоритм ГОСТ Р 34.11-2012, а также модифицировал его, чтобы количество раундов было настраиваемым параметром.

Пример запуска программы для строки *"cvetok"* и с одним раундом:

###### Я провел дифференциальный криптоанализ к реализованной хеш-функции. Он основан на анализе пар сообщений, между которыми существует определенная разность. Я произвожу хеширование с количеством раундов от 0 до 50 с шагом 5

###### График полученных результатов



# Выводы

###### В данной лабораторной работе я познакомился с алгоритмом ГОСТ Р 34.11-2012, реализовал его и провел криптоанализ алгоритма. С увеличением количества раундов количество измененных символов остаётся приблизительно одинаковым, что свидетельствует о том, что алгоритм остаётся стабильно устойчивым к криптоанализу. Например, при 5 раундах количество измененных символов

###### составляет 64, при 25 — 60, а при 50 раундах 57.

###### Из-за недостаточного числа проведенных тестов сложно сделать окончательные выводы о важности полученных результатов.

# Листинг программы

#### CryptoKp.py

import sys  
sys.path.append('D:\PyProjects\pythonProject1\.venv\bobik.py')  
  
from bobik import GOST34112012  
  
def main():  
 iterations = 50  
 m = GOST34112012(digest\_size=32)  
 m.update("cvetok")  
 d = m.hexdigest()  
 counter = 0  
 temp = d  
 for i in range(iterations-1):  
 m = GOST34112012(digest\_size=32)  
 m.update(d)  
 d = m.hexdigest()  
 if (i+2)%5 == 0:  
 for j in range(len(d)):  
 if d[j] != temp[j]:  
 counter +=1  
 print(i+2 ,"\_\_\_\_\_\_", counter)  
 counter = 0  
 print(m.hexdigest())  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 main()

#### bobik.py

# coding: utf-8  
# PyGOST -- Pure Python GOST cryptographic functions library  
# Copyright (C) 2015-2020 Sergey Matveev <stargrave@stargrave.org>  
#  
# This program is free software: you can redistribute it and/or modify  
# it under the terms of the GNU General Public License as published by  
# the Free Software Foundation, version 3 of the License.  
#  
# This program is distributed in the hope that it will be useful,  
# but WITHOUT ANY WARRANTY; without even the implied warranty of  
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the  
# GNU General Public License for more details.  
#  
# You should have received a copy of the GNU General Public License  
# along with this program. If not, see <http://www.gnu.org/licenses/>.  
*""" GOST R 34.11-2012 (Streebog) hash function common files  
  
This is implementation of :rfc:`6986`. Most function and variable names are  
taken according to specification's terminology.  
"""*import sys  
sys.path.append('D:\PyProjects\pythonProject1\.venv\Lib\site-packages\pygost-master')  
  
from copy import copy  
from struct import pack  
from struct import unpack  
  
from pygost.iface import PEP247  
from pygost.utils import hexdec  
from pygost.utils import strxor  
from pygost.utils import xrange  
  
  
BLOCKSIZE = 64  
Pi = bytearray((  
 252, 238, 221, 17, 207, 110, 49, 22, 251, 196, 250,  
 218, 35, 197, 4, 77, 233, 119, 240, 219, 147, 46,  
 153, 186, 23, 54, 241, 187, 20, 205, 95, 193, 249,  
 24, 101, 90, 226, 92, 239, 33, 129, 28, 60, 66,  
 139, 1, 142, 79, 5, 132, 2, 174, 227, 106, 143,  
 160, 6, 11, 237, 152, 127, 212, 211, 31, 235, 52,  
 44, 81, 234, 200, 72, 171, 242, 42, 104, 162, 253,  
 58, 206, 204, 181, 112, 14, 86, 8, 12, 118, 18,  
 191, 114, 19, 71, 156, 183, 93, 135, 21, 161, 150,  
 41, 16, 123, 154, 199, 243, 145, 120, 111, 157, 158,  
 178, 177, 50, 117, 25, 61, 255, 53, 138, 126, 109,  
 84, 198, 128, 195, 189, 13, 87, 223, 245, 36, 169,  
 62, 168, 67, 201, 215, 121, 214, 246, 124, 34, 185,  
 3, 224, 15, 236, 222, 122, 148, 176, 188, 220, 232,  
 40, 80, 78, 51, 10, 74, 167, 151, 96, 115, 30,  
 0, 98, 68, 26, 184, 56, 130, 100, 159, 38, 65,  
 173, 69, 70, 146, 39, 94, 85, 47, 140, 163, 165,  
 125, 105, 213, 149, 59, 7, 88, 179, 64, 134, 172,  
 29, 247, 48, 55, 107, 228, 136, 217, 231, 137, 225,  
 27, 131, 73, 76, 63, 248, 254, 141, 83, 170, 144,  
 202, 216, 133, 97, 32, 113, 103, 164, 45, 43, 9,  
 91, 203, 155, 37, 208, 190, 229, 108, 82, 89, 166,  
 116, 210, 230, 244, 180, 192, 209, 102, 175, 194, 57,  
 75, 99, 182,  
))  
  
A = [unpack(">Q", hexdec(s))[0] for s in (  
 "8e20faa72ba0b470", "47107ddd9b505a38", "ad08b0e0c3282d1c", "d8045870ef14980e",  
 "6c022c38f90a4c07", "3601161cf205268d", "1b8e0b0e798c13c8", "83478b07b2468764",  
 "a011d380818e8f40", "5086e740ce47c920", "2843fd2067adea10", "14aff010bdd87508",  
 "0ad97808d06cb404", "05e23c0468365a02", "8c711e02341b2d01", "46b60f011a83988e",  
 "90dab52a387ae76f", "486dd4151c3dfdb9", "24b86a840e90f0d2", "125c354207487869",  
 "092e94218d243cba", "8a174a9ec8121e5d", "4585254f64090fa0", "accc9ca9328a8950",  
 "9d4df05d5f661451", "c0a878a0a1330aa6", "60543c50de970553", "302a1e286fc58ca7",  
 "18150f14b9ec46dd", "0c84890ad27623e0", "0642ca05693b9f70", "0321658cba93c138",  
 "86275df09ce8aaa8", "439da0784e745554", "afc0503c273aa42a", "d960281e9d1d5215",  
 "e230140fc0802984", "71180a8960409a42", "b60c05ca30204d21", "5b068c651810a89e",  
 "456c34887a3805b9", "ac361a443d1c8cd2", "561b0d22900e4669", "2b838811480723ba",  
 "9bcf4486248d9f5d", "c3e9224312c8c1a0", "effa11af0964ee50", "f97d86d98a327728",  
 "e4fa2054a80b329c", "727d102a548b194e", "39b008152acb8227", "9258048415eb419d",  
 "492c024284fbaec0", "aa16012142f35760", "550b8e9e21f7a530", "a48b474f9ef5dc18",  
 "70a6a56e2440598e", "3853dc371220a247", "1ca76e95091051ad", "0edd37c48a08a6d8",  
 "07e095624504536c", "8d70c431ac02a736", "c83862965601dd1b", "641c314b2b8ee083",  
)]  
  
Tau = (  
 0, 8, 16, 24, 32, 40, 48, 56,  
 1, 9, 17, 25, 33, 41, 49, 57,  
 2, 10, 18, 26, 34, 42, 50, 58,  
 3, 11, 19, 27, 35, 43, 51, 59,  
 4, 12, 20, 28, 36, 44, 52, 60,  
 5, 13, 21, 29, 37, 45, 53, 61,  
 6, 14, 22, 30, 38, 46, 54, 62,  
 7, 15, 23, 31, 39, 47, 55, 63,  
)  
  
C = [hexdec("".join(s))[::-1] for s in (  
 (  
 "b1085bda1ecadae9ebcb2f81c0657c1f",  
 "2f6a76432e45d016714eb88d7585c4fc",  
 "4b7ce09192676901a2422a08a460d315",  
 "05767436cc744d23dd806559f2a64507",  
 ),  
 (  
 "6fa3b58aa99d2f1a4fe39d460f70b5d7",  
 "f3feea720a232b9861d55e0f16b50131",  
 "9ab5176b12d699585cb561c2db0aa7ca",  
 "55dda21bd7cbcd56e679047021b19bb7",  
 ),  
 (  
 "f574dcac2bce2fc70a39fc286a3d8435",  
 "06f15e5f529c1f8bf2ea7514b1297b7b",  
 "d3e20fe490359eb1c1c93a376062db09",  
 "c2b6f443867adb31991e96f50aba0ab2",  
 ),  
 (  
 "ef1fdfb3e81566d2f948e1a05d71e4dd",  
 "488e857e335c3c7d9d721cad685e353f",  
 "a9d72c82ed03d675d8b71333935203be",  
 "3453eaa193e837f1220cbebc84e3d12e",  
 ),  
 (  
 "4bea6bacad4747999a3f410c6ca92363",  
 "7f151c1f1686104a359e35d7800fffbd",  
 "bfcd1747253af5a3dfff00b723271a16",  
 "7a56a27ea9ea63f5601758fd7c6cfe57",  
 ),  
 (  
 "ae4faeae1d3ad3d96fa4c33b7a3039c0",  
 "2d66c4f95142a46c187f9ab49af08ec6",  
 "cffaa6b71c9ab7b40af21f66c2bec6b6",  
 "bf71c57236904f35fa68407a46647d6e",  
 ),  
 (  
 "f4c70e16eeaac5ec51ac86febf240954",  
 "399ec6c7e6bf87c9d3473e33197a93c9",  
 "0992abc52d822c3706476983284a0504",  
 "3517454ca23c4af38886564d3a14d493",  
 ),  
 (  
 "9b1f5b424d93c9a703e7aa020c6e4141",  
 "4eb7f8719c36de1e89b4443b4ddbc49a",  
 "f4892bcb929b069069d18d2bd1a5c42f",  
 "36acc2355951a8d9a47f0dd4bf02e71e",  
 ),  
 (  
 "378f5a541631229b944c9ad8ec165fde",  
 "3a7d3a1b258942243cd955b7e00d0984",  
 "800a440bdbb2ceb17b2b8a9aa6079c54",  
 "0e38dc92cb1f2a607261445183235adb",  
 ),  
 (  
 "abbedea680056f52382ae548b2e4f3f3",  
 "8941e71cff8a78db1fffe18a1b336103",  
 "9fe76702af69334b7a1e6c303b7652f4",  
 "3698fad1153bb6c374b4c7fb98459ced",  
 ),  
 (  
 "7bcd9ed0efc889fb3002c6cd635afe94",  
 "d8fa6bbbebab07612001802114846679",  
 "8a1d71efea48b9caefbacd1d7d476e98",  
 "dea2594ac06fd85d6bcaa4cd81f32d1b",  
 ),  
 (  
 "378ee767f11631bad21380b00449b17a",  
 "cda43c32bcdf1d77f82012d430219f9b",  
 "5d80ef9d1891cc86e71da4aa88e12852",  
 "faf417d5d9b21b9948bc924af11bd720",  
 ),  
)]  
  
  
def add512bit(a, b):  
 *""" Add two 512 integers  
 """* a = bytearray(a)  
 b = bytearray(b)  
 cb = 0  
 res = bytearray(64)  
 for i in range(64):  
 cb = a[i] + b[i] + (cb >> 8)  
 res[i] = cb & 0xff  
 return res  
  
  
def g(n, hsh, msg):  
 res = E(LPS(strxor(hsh[:8], pack("<Q", n)) + hsh[8:]), msg)  
 return strxor(strxor(res, hsh), msg)  
  
  
def E(k, msg):  
 for i in range(12):  
 msg = LPS(strxor(k, msg))  
 k = LPS(strxor(k, C[i]))  
 return strxor(k, msg)  
  
  
def LPS(data):  
 return L(PS(bytearray(data)))  
  
  
def PS(data):  
 res = bytearray(BLOCKSIZE)  
 for i in range(BLOCKSIZE):  
 res[Tau[i]] = Pi[data[i]]  
 return res  
  
  
def L(data):  
 res = []  
 for i in range(8):  
 val = unpack("<Q", data[i \* 8:i \* 8 + 8])[0]  
 res64 = 0  
 for j in range(BLOCKSIZE):  
 if val & 0x8000000000000000:  
 res64 ^= A[j]  
 val <<= 1  
 res.append(pack("<Q", res64))  
 return b"".join(res)  
  
  
class GOST34112012(PEP247):  
 *""" GOST 34.11-2012 big-endian hash  
  
 >>> m = GOST34112012(digest\_size=32)  
 >>> m.update("cve")  
 >>> m.update("tok")  
 >>> m.hexdigest()  
 'e3c9fd89226d93b489a9fe27d686806e24a514e3787bca053c698ec4616ceb78'  
 """* block\_size = BLOCKSIZE  
  
 def \_\_init\_\_(self, data=b"", digest\_size=64):  
 *"""  
 :param digest\_size: hash digest size to compute  
 :type digest\_size: 32 or 64 bytes  
 """* self.data = data  
 self.\_digest\_size = digest\_size  
  
 def copy(self):  
 return GOST34112012(copy(self.data), self.digest\_size)  
  
 @property  
 def digest\_size(self):  
 return self.\_digest\_size  
  
 def update(self, data):  
 *""" Append data that has to be hashed  
 """* self.data += data.encode('utf-8')  
  
 def digest(self):  
 *""" Get hash of the provided data  
 """* hsh = BLOCKSIZE \* (b"\x01" if self.digest\_size == 32 else b"\x00")  
 chk = bytearray(BLOCKSIZE \* b"\x00")  
 n = 0  
 data = self.data  
 for i in xrange(0, len(data) // BLOCKSIZE \* BLOCKSIZE, BLOCKSIZE):  
 block = data[i:i + BLOCKSIZE]  
 hsh = g(n, hsh, block)  
 chk = add512bit(chk, block)  
 n += 512  
  
 # Padding  
 padblock\_size = len(data) \* 8 - n  
 data += b"\x01"  
 padlen = BLOCKSIZE - len(data) % BLOCKSIZE  
 if padlen != BLOCKSIZE:  
 data += b"\x00" \* padlen  
  
 hsh = g(n, hsh, data[-BLOCKSIZE:])  
 n += padblock\_size  
 chk = add512bit(chk, data[-BLOCKSIZE:])  
 hsh = g(0, hsh, pack("<Q", n) + 56 \* b"\x00")  
 hsh = g(0, hsh, chk)  
 return hsh[-self.\_digest\_size:]

**7 Список используемой литературы**

###### Хэш-функция, что это такое? https://habr.com/ru/articles/534596/

##### [Стрибог (хеш-функция) — Википедия (wikipedia.org)](https://ru.wikipedia.org/wiki/Стрибог_(хеш-функция))

##### Репозиторий с зеркалом PyGost, отсюда брал реализацию алгоритма, но она работает с ошибками, поэтому Bobik.py представляет собой исправленную версию алгоритма [mosquito/pygost: pygost mirror form http://pygost.cypherpunks.ru/ (github.com)](https://github.com/mosquito/pygost/tree/master)