Цель работы

Разработать программную реализацию ГОСТ 34.11-2012 «Криптографическая защита информации. Функция хэширования».

Задачи работы

- провести анализ нормативного документа ГОСТ 34.11-2012;
- на основе проведённого анализа разработать описание алгоритма;
- разработать программную реализацию алгоритма;
- проверить работоспособность алгоритма.

Ход работы

Функция хэширования, описанная в ГОСТ 34.11-2012 имеет неофициальное название «Стрибог».

Хэш-функция «Стрибог» может иметь две реализации с результирующим значением длиной 256 или 512 бит. На вход функции подается сообщение, для которого необходимо вычислить хэш-сумму. Если длина сообщения больше 512 бит (или 64 байт), то оно делится на блоки по 512 бит, а оставшийся кусочек дополняется нулями с одной единичкой до 512 бит (или до 64 байт). Если длина сообщения меньше 512 бит, то оно сразу дополняется нулями с единичкой до полных 512 бит.

Основу хеш-функции «Стрибог» составляет функция сжатия (g-функция), построенная на блочном шифре, построенном с помощью конструкции Миягучи — Пренеля, признанной одной из наиболее стойких.

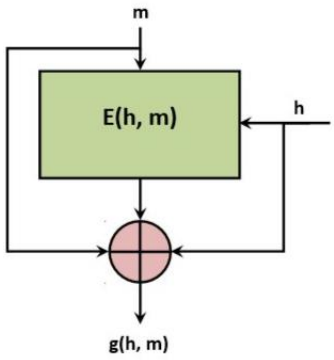

Рисунок 1 - Блочный шифр в режиме Миягучи – Пренел

В целом хэширование производится в три этапа (рисунок 2). Первый этап — инициализация всех нужных параметров, второй этап представляет собой так называемую итерационную конструкцию Меркла — Дамгорда с процедурой МД-усиления, третий этап — завершающее преобразование: функция сжатия применяется к сумме всех блоков сообщения и дополнительно хэшируется длина сообщения и его контрольная сумма.

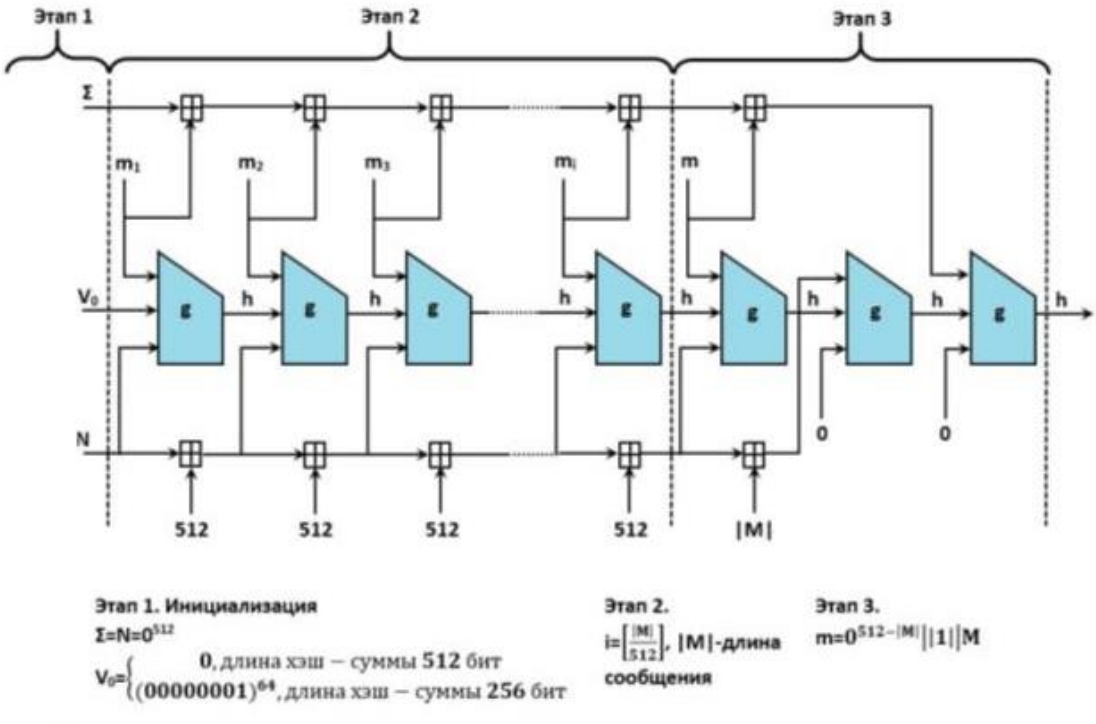

Рисунок 2 - общая схема вычисления хэш-суммы по ГОСТ 34.11-201

Для программной реализации данного алгоритма шифрования и дешифрования использовался язык программирования Python 3.9.

Пример работы разработанного алгоритма:

На рисунке 3 представлен вывод тестового сообщения программы

```
"D:\242220\OneDrive - ITMO UNIVERSITY\4 курс работы\2 семестр\Стандарты криптографии\ГОСТ 34.11-2012\venv\Scripts\python.exe" "D:/242220/OneDrive

--------- Хэш Стрибог (ГОСТ 34.11-2012) ------------

Искомая строка: s7ads6z6MyV2VwAnRSm7bD8oJ4dnXkEenR2tnQXBg98o44TLdP9U2wXUTA89ZVXUsFmwh4DNcyeebmh5qBkkyFdXNm6Amm8J4nfGVbFaXzRMNPkiJAGsKkoWDQDKZERu
Длина хэша: 64 байта
Хэш: 8d5c8557f43d32a1c63980fda66e32630c32723bb90ae09b37ab8f67f252b8503d6a9a0a2127d9f72bee66e6e4b0aed50a07709caf154439f890df291c116f54
Завершено за: 0.09599900245666504 секунд
Длина хэша: 32 байта
Хэш: 546ca6a2266cf303c4e115137a1ef27aa3934c6b2aa7d306e4fc26b5f09a1692
Завершено за: 0.19900226593017578 секунд

Process finished with exit code 0
```

Рисунок 3 - Вывод программной реализации функции хэширования

Для проведения экспериментов был выбрана строка достаточно большой длины. При длине хэша в 64 байта, на его вычисление потребовалось 0.096 секунд.

При вводе той же строки, но длине хэша в 32 байта, на вычисление его ушло 0.2 секунды.

Выводы

В результате выполнения лабораторной работы был проведен анализ документа ГОСТ 34.11-2012, определено описание самой функции хэширования. Помимо этого, был успешно программно реализована данная функция и проведены эксперименты с демонстрацией успешного хэширования.

Приложение А. Исходный код разработанного алгоритма.

```
import json
import re
import time

with open("initial_data.json", 'r', encoding='utf-8') as f:
    initial_data = json.load(f)

SIZE = 64

def bytes2bin(_byte):
    return f"{_byte:08b}"

def _chunks(block, n):
    return (block[i:i + n] for i in range(0, len(block), n))

def add(var1, var2):
    return [(int(x) ^ int(y)) for x, y in zip(var1, var2)]

def L(var1):
    current_var = list(_chunks(var1, SIZE))
    result = []
    for block in current_var:
        temp = 0
        current_block = int(''.join(str(item) for item in block), 2)
        for idx in range(SIZE):
            if current_block & 0x8000000000000000:
                temp ^= int(initial_data["A"][idx], 16)
            current_block <<= 1
        temp = f'{temp:08b}'
        if len(temp) < SIZE:
            temp = list('0' * (SIZE - len(temp)) + temp)
        result.append(temp)
    # Должен возвращать массив в 512 бит
    return [item for sublist in result for item in sublist]

def P(var1):
    result = [[]] * SIZE
    current_var = list(_chunks(var1, 8))
    for i in range(SIZE):
        result[i] = current_var[initial_data["Tau"][i]]
    return [item for sublist in result for item in sublist]

def S(var1):
    result = [[]] * SIZE
    current_var = list(_chunks(var1, 8))
    for i in range(SIZE):
        temp = int(''.join(str(item) for item in current_var[i]), 2)
        try:
            result[i] = [int(char) for char in
f"{initial_data['pi'][temp]:08b}"]
        except Exception as e:
            print(e)
            exit()
    return [item for sublist in result for item in sublist]

def LPS(var1):
    return L(P(S(var1)))

def E(msg, key):
    for i in range(12):
        msg = LPS(add(msg, key))
```

```
            key = LPS(add(msg, f"{int(''.join(initial_data['C'][i]), 16):08b}"))
        return add(msg, key)

    def g(n, _hash, msg):
        result = E(msg, LPS(add(_hash[:8], f'{n:08b}') + _hash[8:]))
        return add(add(result, _hash), msg)

    def get_hash(data, hash_size=256):
        data = data.encode('utf-8')
        _hash = [''.join(bytes2bin(i) for i in b'\x01') if hash_size == 256 else
    ''.join(
            bytes2bin(i) for i in b'\x00')] * SIZE
        _hash = [item for sublist in _hash for item in sublist]
        n = 0
        _sum = [''.join(bytes2bin(i) for i in b'\x00')] * SIZE
        _sum = [item for sublist in _sum for item in sublist]
        bin_data = ''.join(f'{i:08b}' for i in data)
        bin_data = list(_chunks(bin_data, 8))
        for i in range(0, len(bin_data) // SIZE * SIZE, SIZE):
            block = [item for sublist in bin_data[i:i + SIZE] for item in sublist]
            _hash = g(n, _hash, block)
            _sum = add(_sum, block)
            n += 512
        # Длина блока в Битах
        block_pad_size = len(data) * 8 - n
        data += b'\x01'
        pad_length = SIZE - len(data) % SIZE
        if pad_length != SIZE:
            data += b'\x00' * pad_length
        bin_data = ''.join(f'{i:08b}' for i in data)
        bin_data = list(_chunks(bin_data, 8))
        _hash = g(n, _hash, [item for sublist in bin_data[-SIZE:] for item in
    sublist])
        n += block_pad_size
        _sum = add(_sum, [item for sublist in bin_data[-SIZE:] for item in sublist])
        new_n = f'{n:08b}'
        if len(new_n) < 16:
            new_n = list('0' * (16 - len(new_n)) + new_n)
        new_n += [item for sublist in [f'{i:08b}' for i in (64 - len(new_n) // 8) *
    b'\x00'] for item in sublist]
        _hash = g(0, _hash, new_n)
        _hash = g(0, _hash, _sum)
        if hash_size == 256:
            return bytes(
                (int(b, 2)) for b in re.split('(........)', ''.join(str(i) for i in
    _hash[:len(_hash) // 2])) if b).hex()
        else:
            return bytes((int(b, 2)) for b in re.split('(........)', ''.join(str(i)
    for i in _hash)) if b).hex()

    print("\n--------- Хэш Стрибог (ГОСТ 34.11-2012) ------------\n")
    start_time = time.time()
    text =
    "s7ads6z6MyV2VwAnRSm7bD8oJ4dnXkEenR2tnQXBg98o44TLdP9U2wXUTA89ZVXUsFmwh4DNcyeebmh
    5qBkkyFdXNm6Amm8J4nfGVbFaXzRMNPkiJAGsKkoWDQDKZERu"
    print(f"Искомая строка: {text}")
    hash_512 = get_hash(text, hash_size=512)
    print(f"Длина хэша: {len(hash_512)//2} байта")
    print(f"Хэш: {hash_512}")
    print(f"Завершено за: {time.time() - start_time} секунд")
    hash_256 = get_hash(text, hash_size=256)
    print(f"Длина хэша: {len(hash_256)//2} байта")
    print(f"Хэш: {hash_256}")
    print(f"Завершено за: {time.time() - start_time} секунд")
```