**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-307Б-21

Студент: А. В. Крючков

Преподаватель: А. В. Борисов

Оценка:

Дата: 09.05.2024

Москва, 2024

**ОГЛАВЛЕНИЕ**

[1](#_gjdgxs) Тема 3

[2](#_30j0zll) Задание 3

[3](#_1fob9te) Получение варианта 4

[4](#_3znysh7) Реализация алгоритма согласно варианту 6

[5](#_2et92p0) Модификация количества раундов алгоритма 7

6Дифференциальный криптоанализ 3

7Графики 3

8Выводы 4

# Тема

Функции хеширования.

# Задание

№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 16-тиричное число, которое в дальнейшем будет номером варианта.

№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.

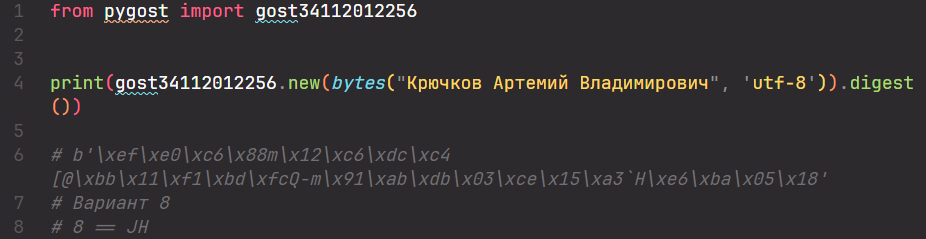
№3. Применить подходы дифференциального криптоанализа к полученным алгоритмам с разным числом раундов.

№4. Построить график зависимости количества раундов и возможности различения отдельных бит при количестве раундов 1,2,3,4, 5....

№5. Сделать выводы

# Получение варианта

Для получения варианта я воспользовался следующей реализацией хеш-функции ГОСТ Р 34.11-2012 (Стрибог):

****

Значит мой вариант 8 = JH

# Реализация алгоритма согласно варианту

Алгоритм JH — это криптографическая хеш-функция, которая была одним из финалистов конкурса NIST на создание нового стандарта хеш-функции SHA-3, хотя и не стала победителем. Суть алгоритма заключается в обеспечении высокой стойкости к криптографическим атакам и эффективности выполнения на современных процессорах, включая те, которые поддерживают векторные инструкции.

Хеш-функция JH состоит из пяти шагов: дополнение сообщения (раздел 4.1), разбор дополненного сообщения на блоки сообщений (раздел 4.2), установка начального значения хеша H(0) (раздел 4.3), вычисление окончательного значения хеша H(N) (раздел 4.4) и генерация дайджеста сообщения путем усечения H(N) (раздел 4.5).

4.1 Дополнение сообщения

Сообщение M дополняется до кратного 512 битам. Предположим, что длина сообщения M составляет ℓ бит. Добавьте бит «1» в конец сообщения, за которым следуют 384 − 1 + (−ℓ mod 512) нулевых битов (для ℓ mod 512 = 0, 1, 2, ..., 510, 511 количество добавляемых нулевых битов составляет 383, 894, 893, ..., 385, 384 соответственно), затем добавьте 128-битный блок, равный числу ℓ, выраженному в двоичной форме в big endian. Таким образом, к сообщению M добавляется как минимум 512 дополнительных бит.

4.2 Разбор дополненного сообщения

После дополнения сообщение разбирается на N 512-битных блоков, M(1), M(2), ..., M(N). 512-битный блок сообщения выражается четырьмя 128-битными словами. Первые 128 бит блока сообщения i обозначаются как M(i)0, следующие 128 бит как M(i)1 и так далее до M(i)3.

4.3 Установка начального значения хеша H(0)

Начальное значение хеша H(0) устанавливается в зависимости от размера дайджеста сообщения. Первые два байта H(−1) устанавливаются как размер дайджеста сообщения, а остальные байты H(−1) устанавливаются как 0. Установите M(0) как 0. Тогда H(0) = F8(H(−1), M(0)).

4.4 Вычисление окончательного значения хеша H(N)

Функция сжатия F8 применяется для генерации H(N) путем сжатия M(1), M(2), ..., M(N) итеративно. Окончательное значение хеша H(N) вычисляется следующим образом:

for i = 1 to N,

H(i) = F8(H(i−1), M(i));

4.5 Генерация дайджеста сообщения

Дайджест сообщения генерируется путем усечения H(N).

- 4.5.1 JH-224: Последние 224 бита H(N) используются как дайджест сообщения JH-224: H(N),800∥H(N),801∥ ... ∥H(N),1023.

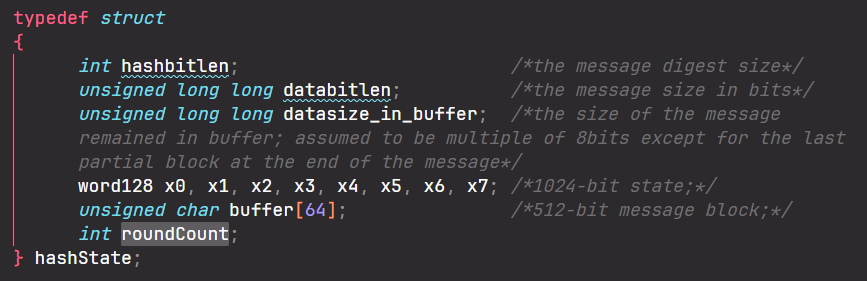
- 4.5.2 JH-256: Последние 256 бит H(N) используются как дайджест сообщения JH-256: H(N),768∥H(N),769∥ ... ∥H(N),1023.

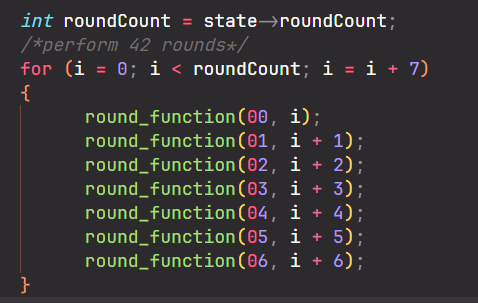
- 4.5.3 JH-384: Последние 384 бита H(N) используются как дайджест сообщения JH-384: H(N),640∥H(N),641∥ ... ∥H(N),1023.

- 4.5.4 JH-512: Последние 512 бит H(N) используются как дайджест сообщения JH-512: H(N),512∥H(N),513∥ ... ∥H(N),1023.

# Модификация количества раундов алгоритма

Вынес количество раундов в структуру для последующего использования.





# Дифференциальный криптоанализ

Дифференциальный криптоанализ — это метод атаки на блочные шифры и хеш-функции, который анализирует, как различия во входных данных могут повлиять на различия в выходных данных после прохождения через криптографическое преобразование.

В рамках исследования была проведена серия экспериментов с целью анализа влияния изменений во входных текстах на результаты шифрования при разном количестве раундов. Основываясь на теоретических предложениях, описанных в статье, были созданы пары входных текстов, отличающихся несколькими битами. Эти пары были зашифрованы с использованием различного числа раундов.

Для реализации эксперимента был разработан программный код на языке C++, который сначала запрашивал у пользователя ввод сообщения, числа битов до которого проверять и числа раундов, до которого проверять.

Для каждого раунда шифрования функция хеширования вызывалась дважды: первый раз с исходным сообщением, а второй — с сообщением, в котором был инвертирован несколько случайно выбранный бит. Результаты каждого хеширования сохранялись, и после этого считалось количество бит, которые отличались в двух хешах.

Результаты эксперимента показали, что изменения в даже одном бите входного текста приводят к значительным изменениям в результатах хеширования, что демонстрирует чувствительность алгоритма к входным данным. Однако от числа раундов зависимость не прослеживалась, вследствие хорошей защиты алгоритма от подобного вида атак.

Код:

*int* calculateHash(*char* \**text*, *int* *roundCount*, *unsigned* *char* \**hashResult*)

{

      hashState state;

      if (Init(&state, 256, roundCount) != SUCCESS)

      {

            printf("Ошибка при инициализации состояния хэша.\n");

            return 1;

      }

      if (Update(&state, (*unsigned* *char* \*)text, strlen(text)) != SUCCESS)

      {

            printf("Ошибка при обновлении хэша.\n");

            return 1;

      }

      if (Final(&state, hashResult) != SUCCESS)

      {

            printf("Ошибка при завершении вычисления хэша.\n");

            return 1;

      }

}

*void* flipBitsInString(*char* \**text*, *int* *size*, *int* *bits*, *char* \**result*)

{

*int* bitIndex = 0;

      for (*int* i = 0; i < size; i++)

      {

            result[i] = text[i];

      }

      for (*int* i = 0; i < size && bitIndex < bits; i++)

      {

            for (*int* j = 0; j < 8 && bitIndex < bits; j++)

            {

                  if ((result[i] & (1 << j)) == 0)

                  {

                        result[i] |= (1 << j);

                        bitIndex++;

                  }

                  else

                  {

                        result[i] &= ~(1 << j);

                        bitIndex++;

                  }

            }

      }

}

*int* countDifferentBits(*const* *char* \**str1*, *const* *char* \**str2*, *int* *size*)

{

*int* count = 0;

      for (*int* i = 0; i < size; i++)

      {

*unsigned* *char* diff = str1[i] ^ str2[i];

            for (*int* j = 0; j < 8; j++)

            {

                  if (diff & (1 << j))

                  {

                        count++;

                  }

            }

      }

      return count;

}

*void* compareHashes(*char* \**text*, *int* *bitsToChange*, *int* *roundCount*) {

*unsigned* *char* originalHash[32];

*unsigned* *char* modifiedHash[32];

*char* modifiedText[strlen(text)];

    for (*int* i = 1; i <= bitsToChange; i++) {

        for (*int* j = 1; j <= roundCount; j++) {

            if (calculateHash(text, j, originalHash) != 0) {

                return;

            }

            flipBitsInString(text, strlen(text), i, modifiedText);

            if (calculateHash(modifiedText, j, modifiedHash) != 0) {

                return;

            }

*int* diffBits = countDifferentBits((*const* *char* \*)originalHash, (*const* *char* \*)modifiedHash, 32);

            printf("%d,%d,%d\n", i, j, diffBits);

        }

    }

}

*int* main()

{

*int* roundCount, bitsToChange;

*char* text[1000];

      printf("Введите количество раундов: ");

      scanf("%d", &roundCount);

      printf("Введите количество битов которые надо менять: ");

      scanf("%d", &bitsToChange);

      printf("Введите текст для хэширования: ");

      scanf(" *%*[^\n]s", text);

      compareHashes(text, bitsToChange, roundCount);

      return 0;

}

Полный код алгоритма и исследования доступен по ссылке:

<https://www3.ntu.edu.sg/home/wuhj/research/jh/jh_sse2_opt64.h>

Протокол работы программы:

Введите количество раундов: 4

Введите количество битов которые надо менять: 4

Введите текст для хэширования: Крючков

1,1,128

1,2,128

1,3,128

1,4,128

2,1,129

2,2,129

2,3,129

2,4,129

3,1,120

3,2,120

3,3,120

3,4,120

4,1,134

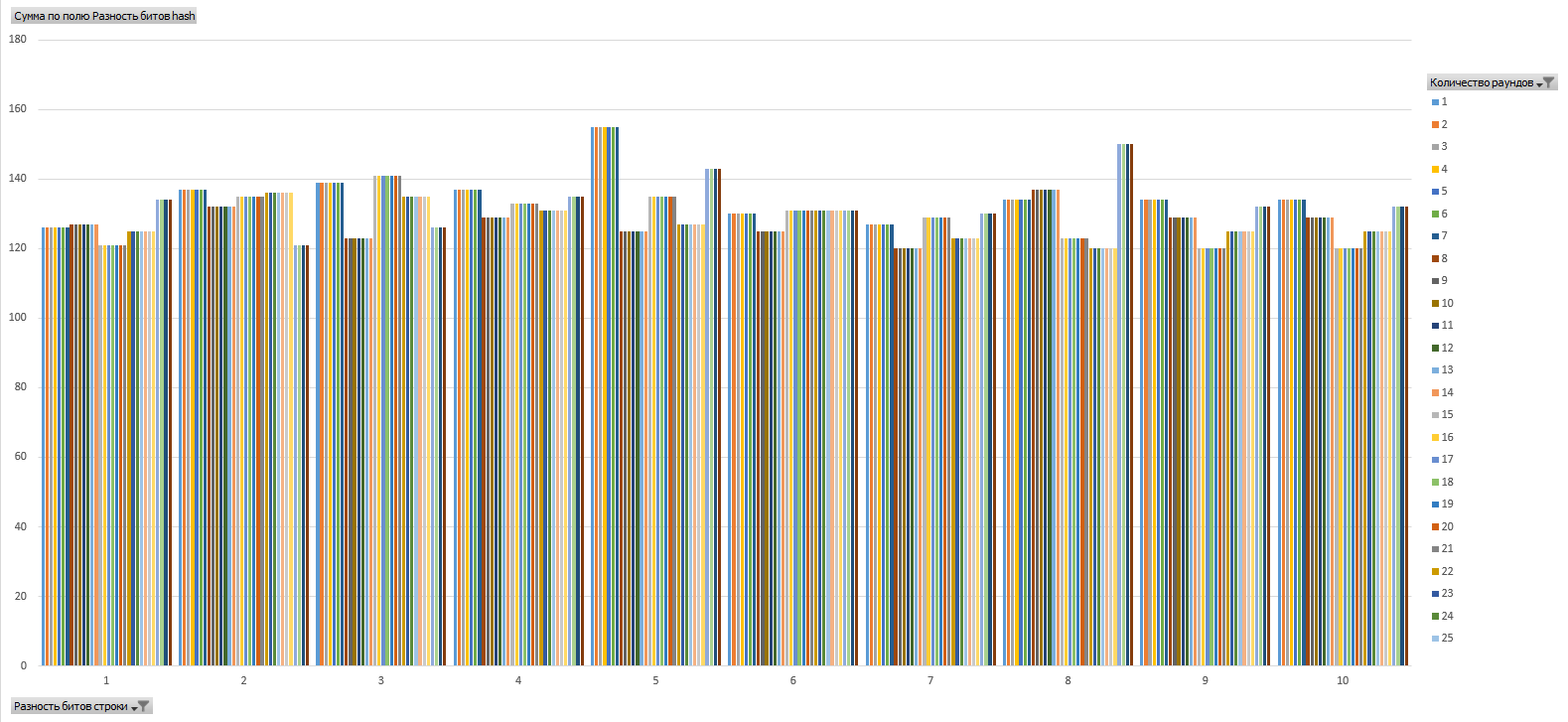
4,2,134

4,3,134

4,4,134

# Графики

На данной диаграмме снизу показано число раундов, а разными цветами обозначено число битов, которые отличаются в строках. Значения гистограмм — это число отличающихся битов в хэшах. Из графика мы видим, что нет какой-либо связи между хэшами при любом числе отличающихся битов и любом числе проведённых раундов сортировки:



# Выводы

Хеш-функция JH использует продвинутую структуру битового среза и оптимизированные операции перестановок и преобразований, повышая тем самым устойчивость алгоритма к различным типам атак, включая дифференциальный криптоанализ.

Основные операции включают дополнение сообщения до необходимой длины, разбиение его на блоки и последовательное применение сжимающих функций, каждая из которых участвует в преобразовании и сжатии данных для получения конечного хеш-значения.

В алгоритме предусмотрена возможность регулировки количества раундов в реализации битового среза, что позволяет настраивать сложность и устойчивость шифра без потери его биективных и пермутационных свойств.

Результаты экспериментов показывают, что хеш-функция JH остается устойчивой к дифференциальному криптоанализу даже при изменении количества раундов, что подчеркивает надежность и гибкость использования различных конфигураций раундов в рамках параметров, установленных разработчиками.

# Список источников

<https://www3.ntu.edu.sg/home/wuhj/research/jh/jh_sse2_opt64.h>