**МИНОБРНАУКИ РОССИИ**
**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ**
**ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**
**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**
**Кафедра МО ЭВМ**

# ОТЧЕТ

**по лабораторной работе №6**

**“Изучение хэш-функций”**

Студент гр. 8382 Мирончик П.Д.

Преподаватель Племянников А.К.

Санкт-Петербург

2021

# ИССЛЕДОВАНИЕ ЛАВИННОГО ЭФФЕКТА MD5, SHA-1, SHA-256, SHA-512

## 1. Задание

1. Открыть текст не менее 1000 знаков. Добавить свое ФИО последней строкой. Перейти к утилите Indiv.Procedures->Hash->Hash Demonstration..

2. Задать хэш-функцию, подлежащую исследованию: MD5, SHA-1, SHA-256, SHA-512.

3. Для каждой хэш-функции повторить следующие действия:

a. Измените (добавлением, заменой, удалением символа) исходный файле

b. Зафиксировать количество измененных битов в дайджесте модифицированного сообщения.

c. Вернуть сообщение в исходное состояние.

4. Выполните процедуру 3 раза (добавлением, заменой, удалением символа) и подсчитайте среднее количество измененных бит дайджеста. Зафиксировать результаты в таблице.

## 2. Основные параметры и обобщенная схема хэш-функций MD5, SHA-1

### 2.1. MD5

MD5 обрабатывает блоки по 512 бит.

В исходную строку дописывают единичный байт 0x80, а затем дописывают нулевые биты, до тех пор, пока длина сообщения не будет сравнима с 448 по модулю 512. Далее в сообщение дописывается 64-битное представление длины исходного сообщения.

Затем выполняется инициализация буфера ABCD значениями, указанными на рис. 1.

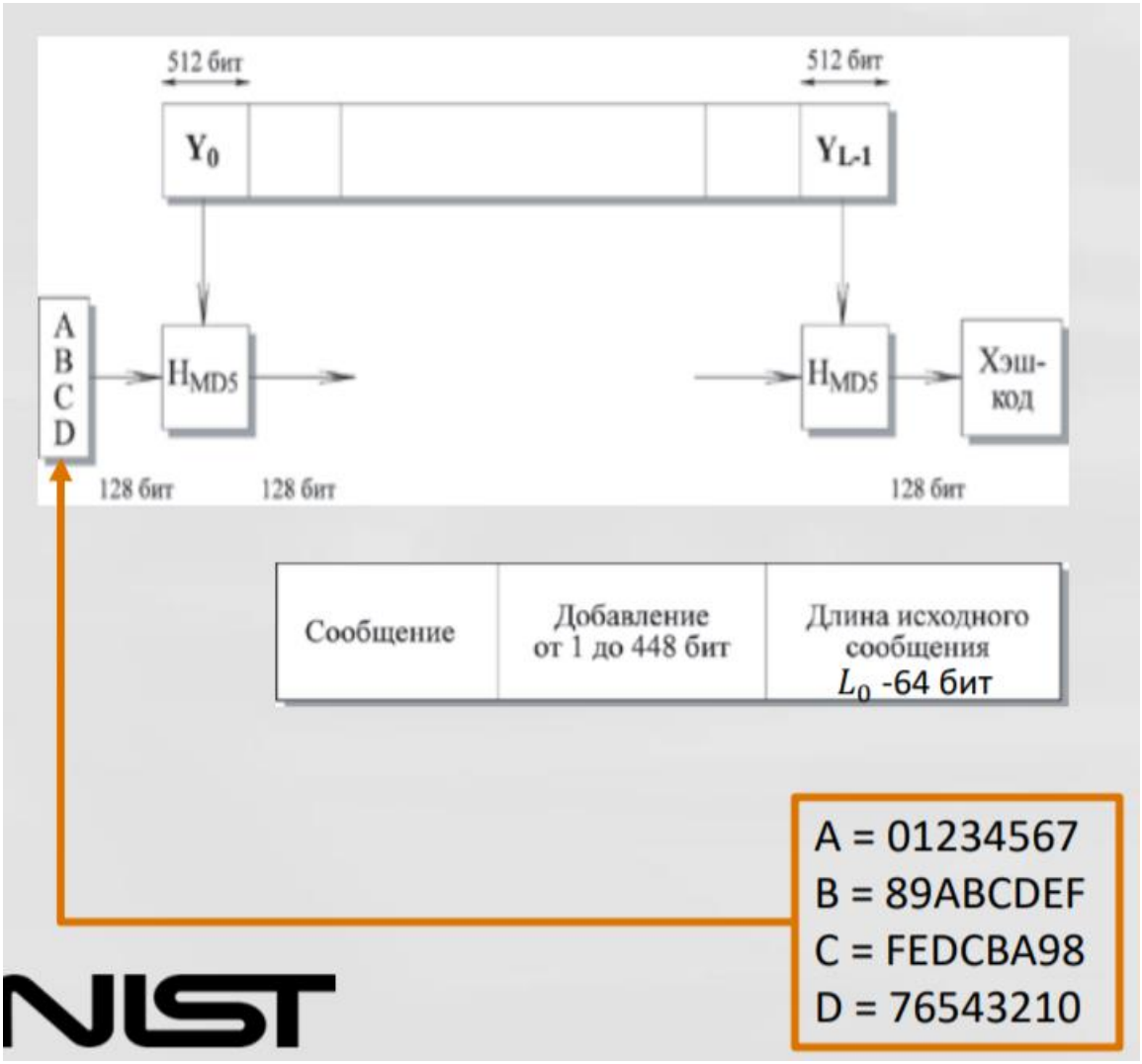


Рис. 1 – Общая схема MD5

Формирование хеша происходит цикличным вычислением хешей блоков. Вычисление блока (функция сжатия) состоит из операций, представленных на рис. 2:

1. Выполнение четырех циклов сжатия.

2. Сложение элементов исходного вектора ABCD и измененного в результате выполнения циклов по модулю $2^{32}$.

После выполнения вычисления хешей всех блоков вектор ABCD будет являться вычисленным исходного текста.

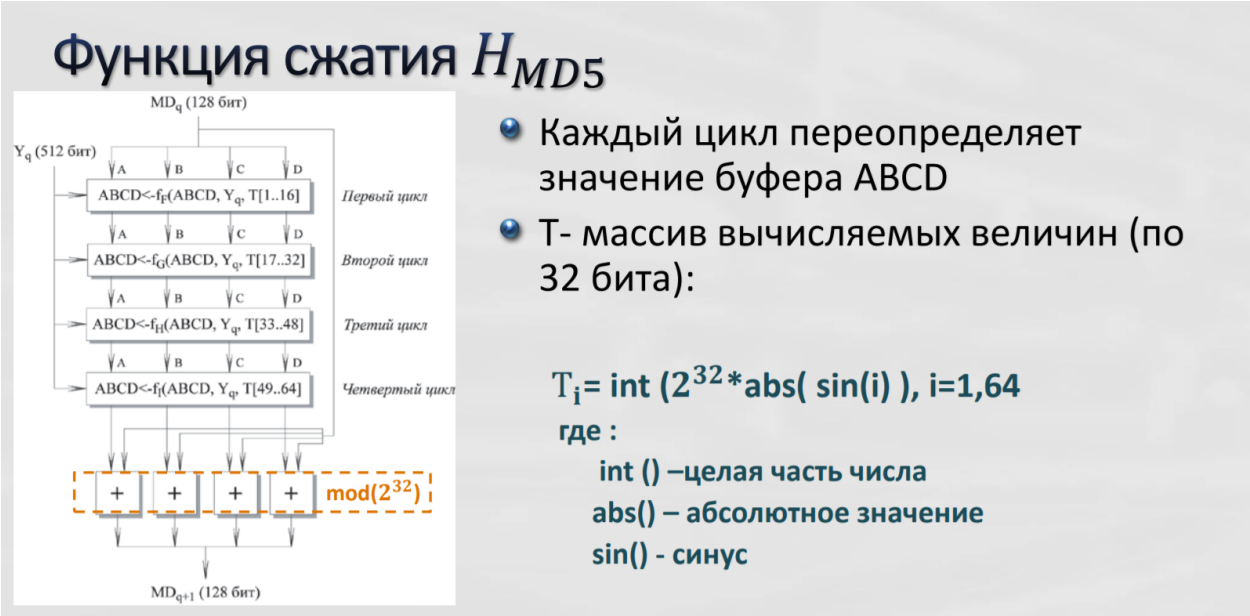


Рис. 2 – Функция сжатия MD5

Рассмотрим алгоритм цикла сжатия, представленный на рис. 3. На вход цикл сжатия принимает 16 32-битных элементов сгенерированного шума T, вектор ABCD, раундовую функцию f и непосредственно сжимаемый блок. Цикл выполняется 16 раз и состоит из 6 шагов:

1. Вычисление значение раундовой функции f с параметрами B, C, D.
2. Сложение с A.
3. Сложение с x[i], где x[i] – k-е 32-битное слово сжимаемого блока исходного текста.
4. Сложение с T[i], где T[i] – i-й элемент шума T.
5. Циклический сдвиг на 5 бит влево.
6. Сложение с B.

Полученный результат записывается в A и буфер ABCD сдвигается на одно слово вправо.

Все операции сложения в алгоритме MD5 выполняются по модулю $2^{32}$.

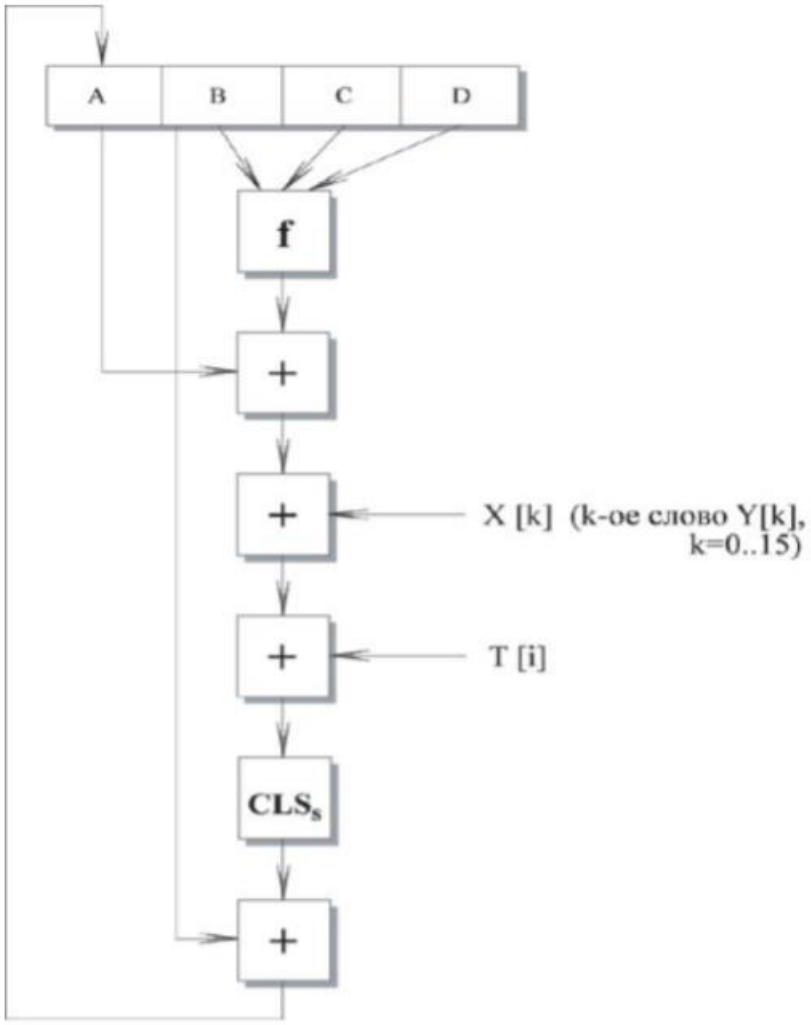


Рис. 3 – Цикл сжатия MD5

MD5 обладает следующими свойствами:

1. Каждый бит хэш-кода является функцией от каждого бита входа.

2. Комплексное повторение элементарных функций обеспечивает хорошее перемешивание результата.

3. MD5 является наиболее сильной хэш-функцией для 128- битного хэш-кода.

### 2.2. SHA-1

SHA-1 достаточно похож на MD5 по своей структуре (рис. 4). Аналогично MD5 происходит выравнивание длины до кратной 512 и обрабатываются блоки по 512 бит, которые проходят цикличный процесс сжатия в функциях сжатия.

После выравнивания длины выполняется инициализация буфера ABCDE значениями, представленными на рис. 4.

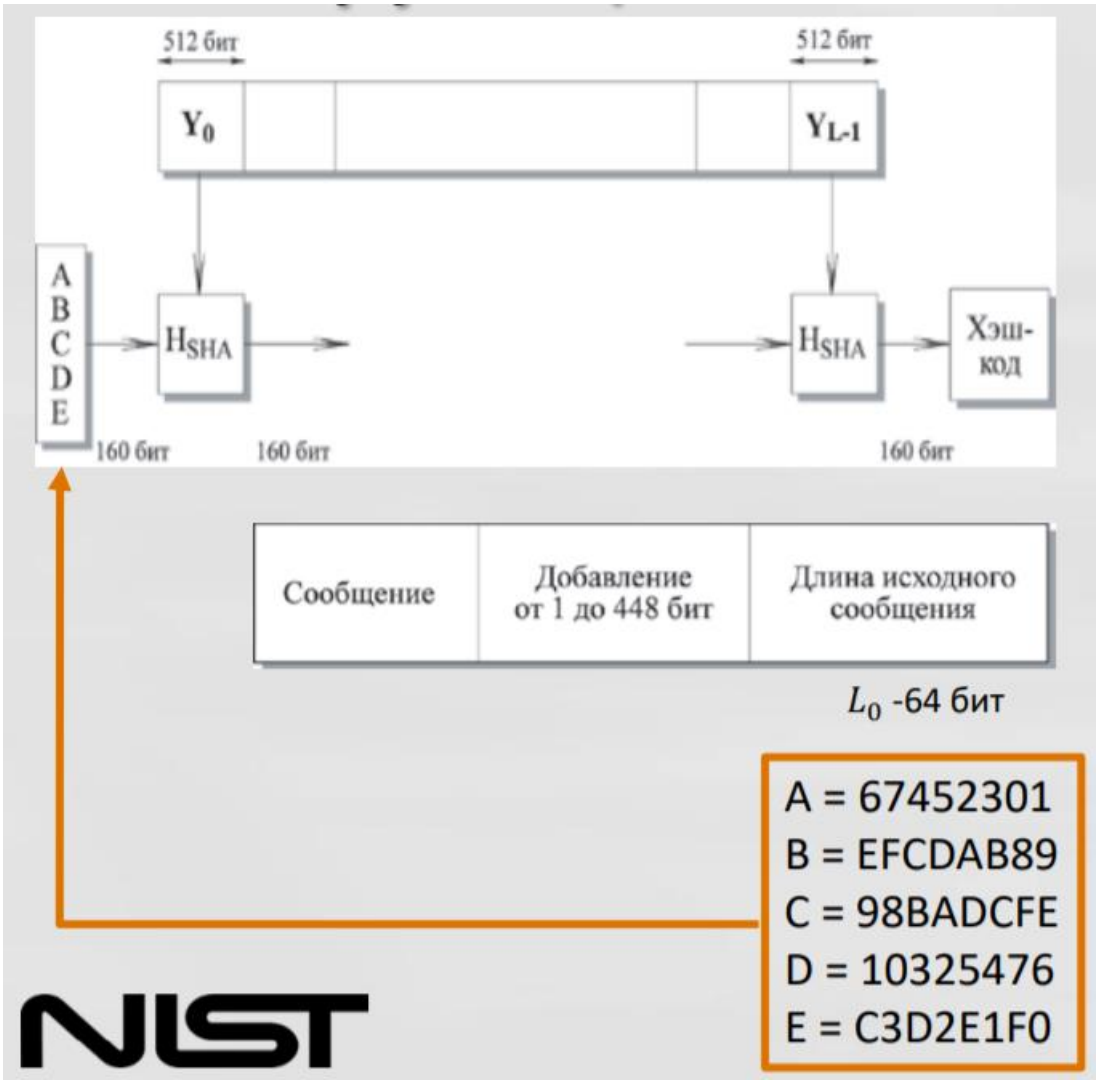


Рис. 4 – Алгоритм SHA-1

Каждый блок обрабатывается функцией сжатия, алгоритм которой представлен на рис. 5.

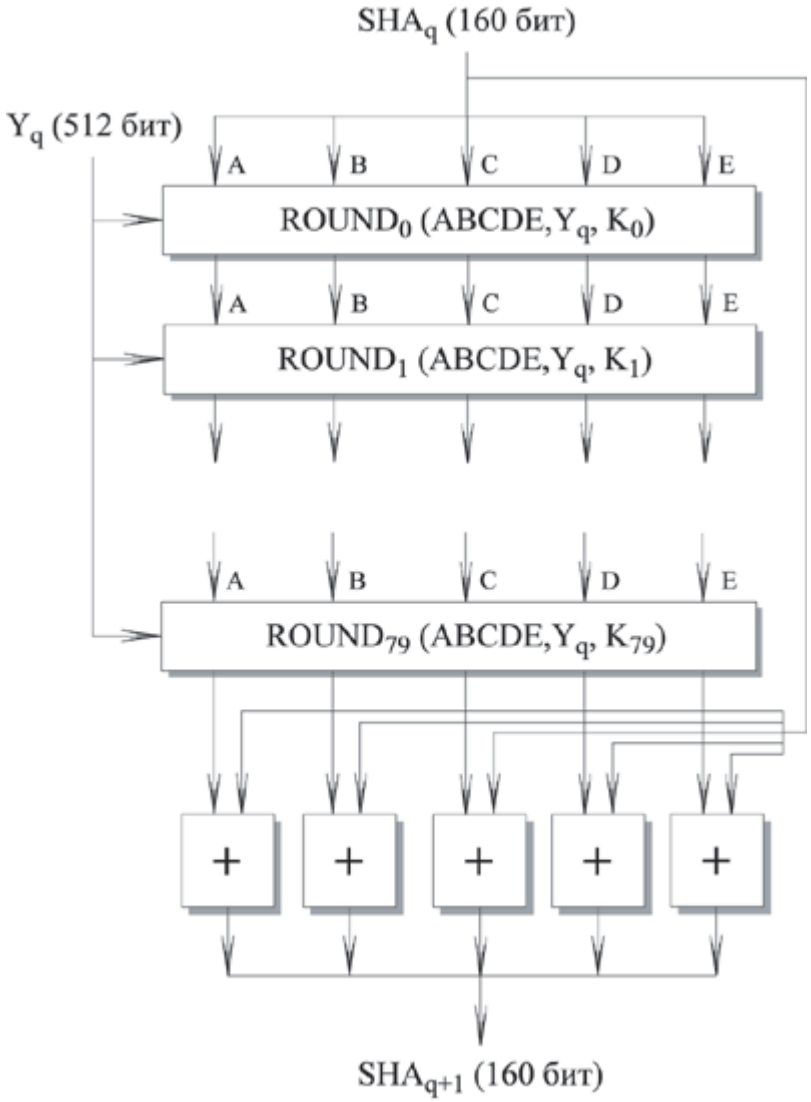


Рис. 5 – Функция сжатия SHA-1

Функция сжатия содержит 80 раундов преобразований вектора ABCDE, после цикличного выполнения которых производится сложение текущего вектора ABCDE и исходным, который был до выполнения этих циклов.

Алгоритм цикла сжатия представлен на рис. 6.

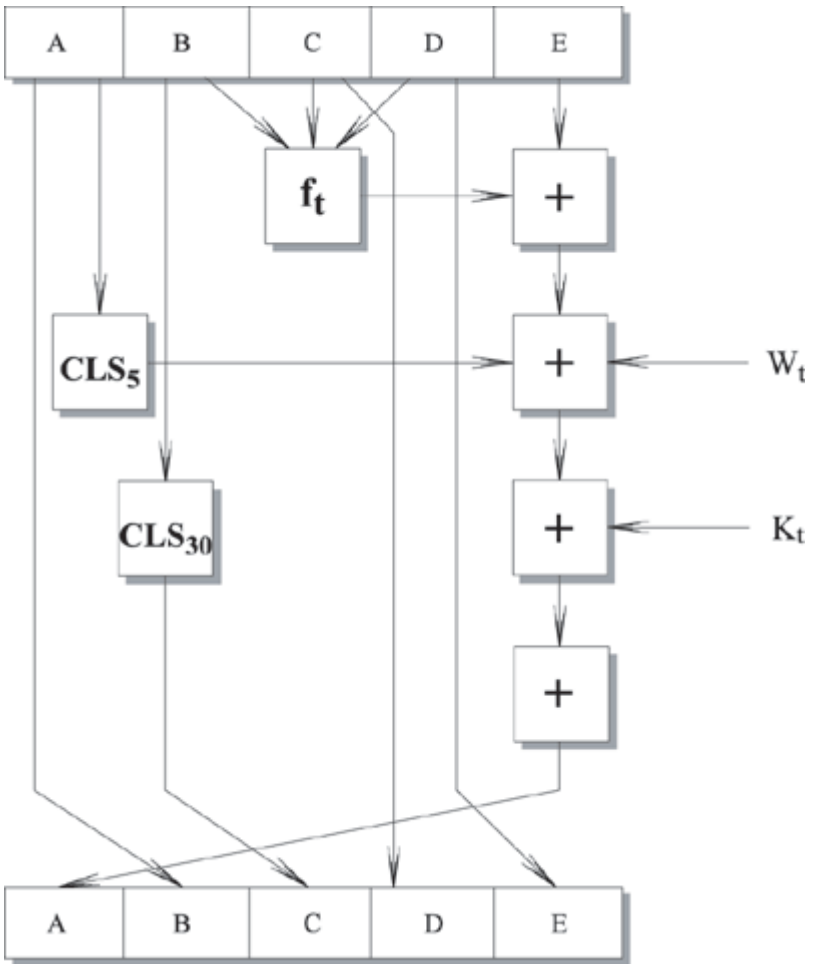


Рис. 6 – Цикл сжатия SHA-1

Цикл сжатия принимает на вход константу $K_t$, сгенерированное слово $W_t$, раундовую функцию $f_t$ и выполняет преобразования, указанные на рис. 6.

Значения констант K описаны ниже:

```
0  <= t <= 19  Kt = 5A827999
20 <= t <= 39  Kt = 6ED9EBA1
40 <= t <= 59  Kt = 8F1BBCDC
60 <= t <= 79  Kt = CA62C1D6
```

Вычисление слов $W_t$ и список раундовых функций представлен на рис. 7 и 8.

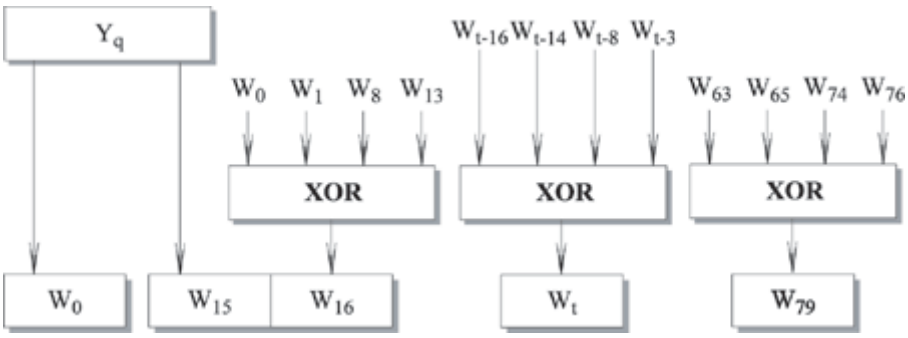


Рис. 7 – Вычисление слова $W_t$

| Номер цикла | ft (B, C, D) |
|---|---|
| (0 <= t <= 19) | $(B \wedge C) \vee (\neg B \wedge D)$ |
| (20 <= t <= 39) | $B \oplus C \oplus D$ |
| (40 <= t <= 59) | $(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$ |
| (60 <= t <= 79) | $B \oplus C \oplus D$ |

Рис. 8 – Раундовые функции

## 3. Исследование хеш-функций

Исходный текст:

```
Mr. and Mrs. Dursley, of number four, Privet Drive, were proud
to say that they were perfectly normal, thank you very much.
They were the last people you'd expect to be involved in
anything strange or mysterious, because they just didn't hold
with such nonsense.

Mr. Dursley was the director of a firm called Grunnings, which
made drills. He was a big, beefy man with hardly any neck,
although he did have a very large mustache. Mrs. Dursley was
thin and blonde and had nearly twice the usual amount of neck,
which came in very useful as she spent so much of her time
craning over garden fences, spying on the neighbors. The
Dursleys had a small son called Dudley and in their opinion
there was no finer boy anywhere.

The Dursleys had everything they wanted, but they also had a
secret, and their greatest fear was that somebody would discover
it. They didn't think they could bear it if anyone found out
about the Potters. Mrs. Potter was Mrs. Dursley's sister, but
they hadn't met for several years; in fact, Mrs.

Dursley pretended she didn't have a sister, because her sister
and her good-for-nothing husband were as unDursleyish as it was
```

```
possible to be. The Dursleys shuddered to think what the
neighbors would say if the Potters arrived in the street. The
Dursleys knew that the Potters had a small son, too, but they
had never even seen him. This boy was another good reason for
keeping the Potters away; they didn't want Dudley mixing with a
child like that.

Mironchik Pavel Denisovich
```

Проведем исследование лавинного эффекта для хеш-функций MD5, SHA-1, SHA-256, SHA-512.

Табл. 1 – Исследование лавинного эффекта

| **Функция** | **Изменение** | **Добавление** | **Удаление** | **Среднее** | **%** |
|---|---|---|---|---|---|
| MD5 | 57 | 59 | 71 | 62 | 48 |
| SHA-1 | 83 | 76 | 88 | 82 | 51 |
| SHA-256 | 133 | 133 | 125 | 130 | 50 |
| SHA-512 | 240 | 270 | 260 | 257 | 50 |

Видно, что все хеш-функции показали примерно одинаковый средний результат по проценту числа измененных бит при изменении исходного текста. Разумеется, все они показали разные значения количества измененных бит, однако это связано с длиной хеша, и с учетом этого параметра функции отработали примерно одинаково успешно.

## 4. Выводы

При исследовании лавинного эффекта были исследованы хеш-функции MD5, SHA-1, SHA-256, SHA-512 и получены результаты по количеству измененных бит хеша при изменении части исходного текста: добавлении, удалении или изменении одного символа. Определено, что, с учетом длины хеша, функции показывают примерно одинаковый результат.

# ХЭШ-ФУНКЦИЯ SHA-3

## 1. Задание

1. Открыть шаблон Keccak Hash (SHA-3) в Cryptool 2
2. В модуле Keccak сделать следующие настройки:
   a. Adjust manually=ON
   b. Keccak version= SHA3-512
3. Загрузить файл из предыдущего задания
4. Запустить проигрывание шаблона в режиме ручного управления:
   a. Сохранить скриншоты преобразований первого раунда
   b. Сохранить скриншот заключительной фазы
   c. Сохранить значение дайджеста
5. Вычислить значения дайджеста для модифицированных текстов из предыдущего задания
6. Подсчитать лавинный эффект с помощью самостоятельно разработанной автоматизированной процедуры

## 2. Основные параметры и обобщенная схема

Алгоритм состоит из двух этапов (Рис. 9):

1. Absorbing (впитывание). На каждом шаге очередной блок сообщения $\boldsymbol{pi}$ длиной r подмешивается к части внутреннего состояния S, которая затем целиком модифицируется функцией f - многораундовой бесключевой псевдослучайной перестановкой.

2. Squeezing (отжатие). Чтобы получить хэш, функция f многократно применяется к состоянию, и на каждом шаге извлекается и сохраняется кусок размера r до тех пор, пока не получим выход Z необходимой длины (путем конкатенации).

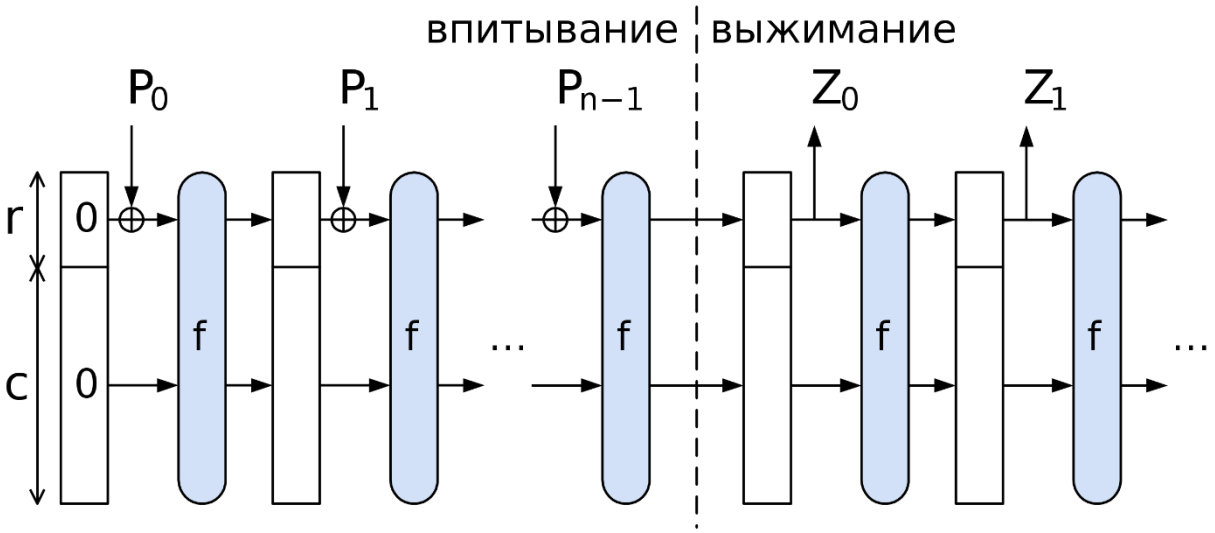


Рис. 9 – Обобщенная схема SHA3

Характеристики алгоритма SHA3.

1. Авторы Keccak доказали, что стойкость такой конструкции неотличима от стойкости идеальной хэш-функции (random oracle model) с размером дайджеста, равным c/2 (всё состояние имеет размер c+r) при условии, что f — идеальная функция перестановки (без повторений внутреннего состояния).

2. Сложность проведения атак гарантируется только до необходимой величины. После увеличения длины выхода хэш-функции за пределы расчетного, стойкость функции перестаёт зависеть от размера выхода.

3. Хэш-функция Keccak реализована таким образом, что функцию перестановки f пользователь может выбирать самостоятельно из набора предопределенных функций {f-25, f-50, f-100, f-200, f-400, f-800, f-1600}.

Далее на рисунках 10-17 представлены шаги, выполняемые внутри раунда функции f, начальная и конечная фазы.

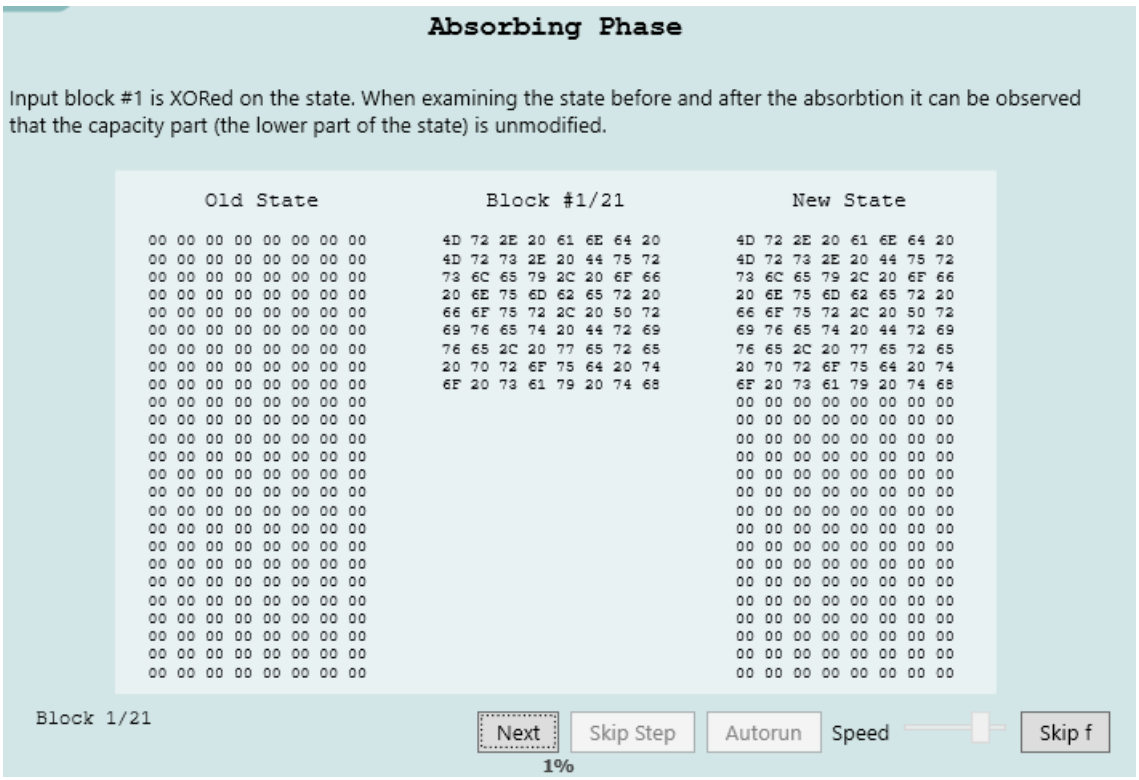


Рис. 10 – Начальная фаза SHA3

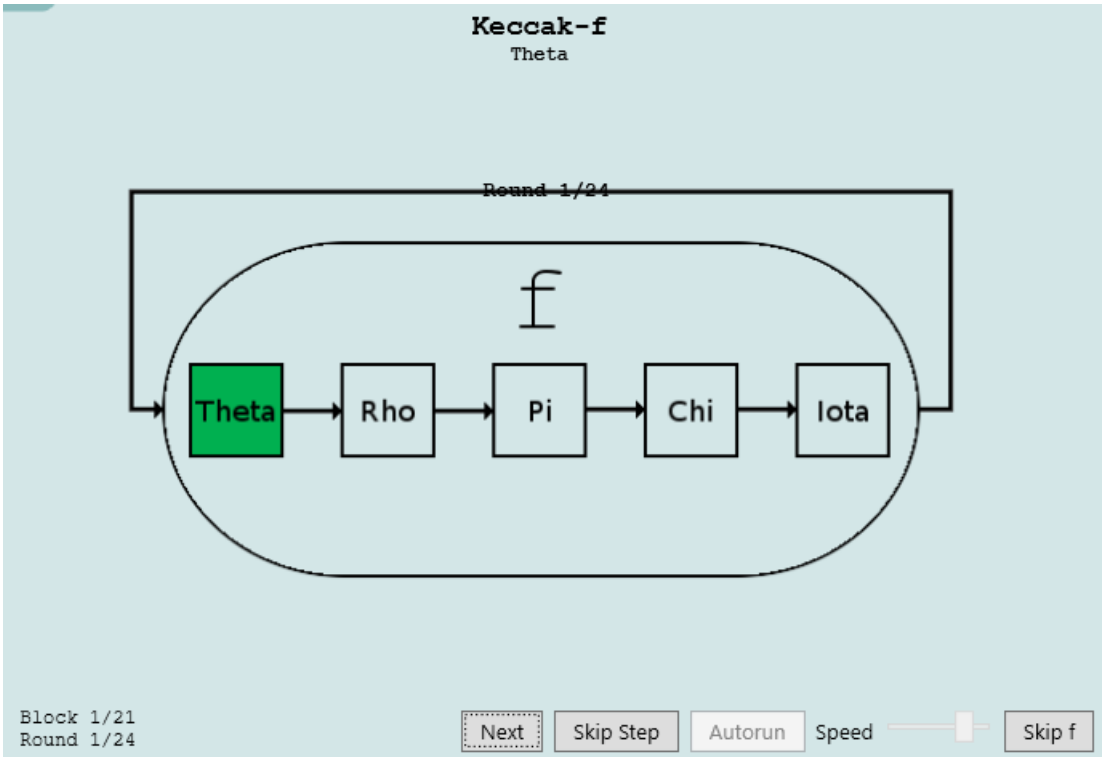


Рис. 11 – Структура функции f

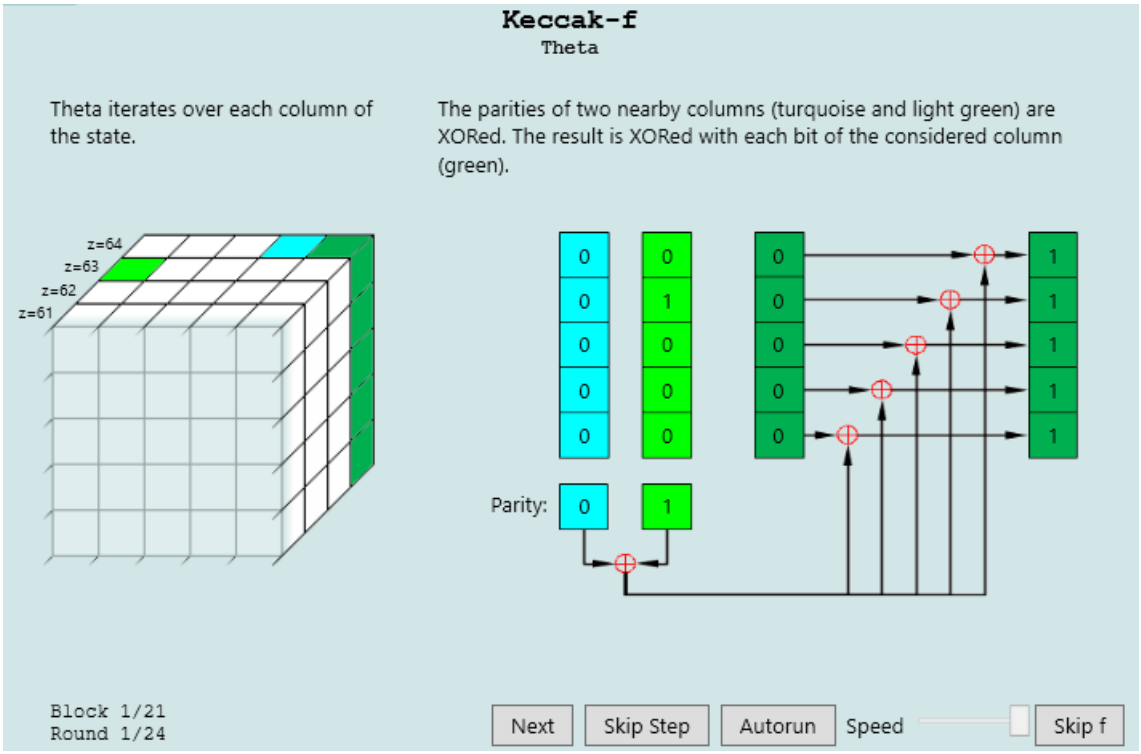


Рис. 12 – Theta операция

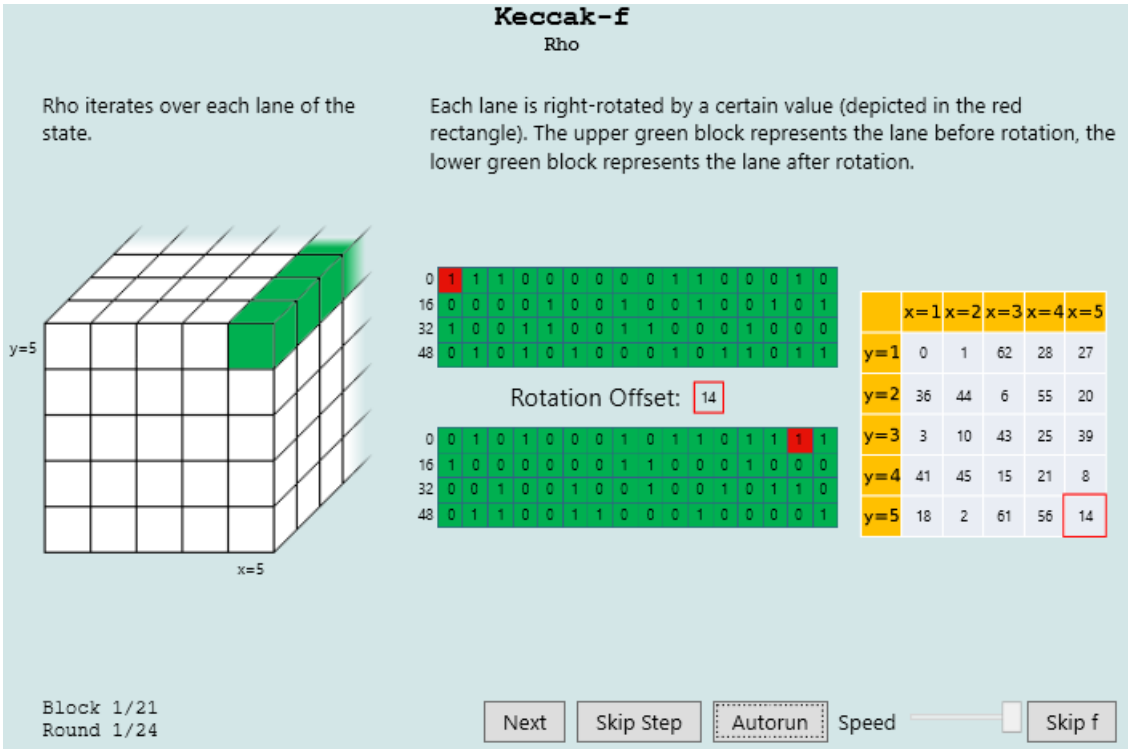


Рис. 13 – Rho операция

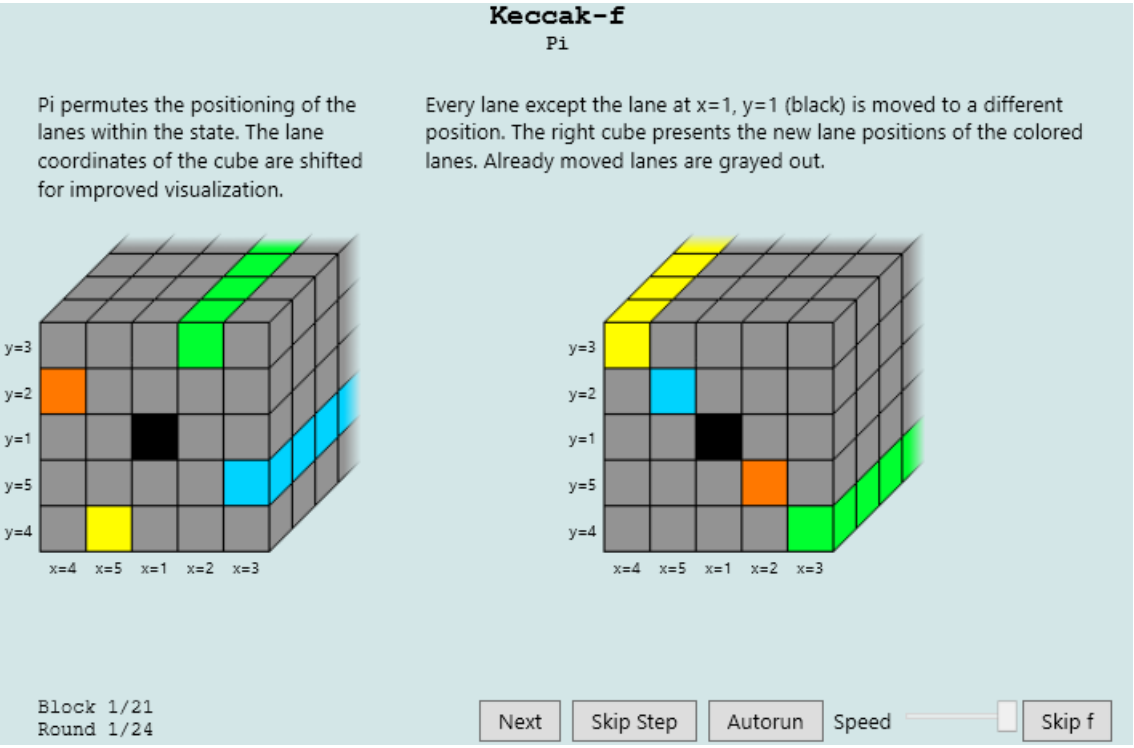


Рис. 14 – Pi операция

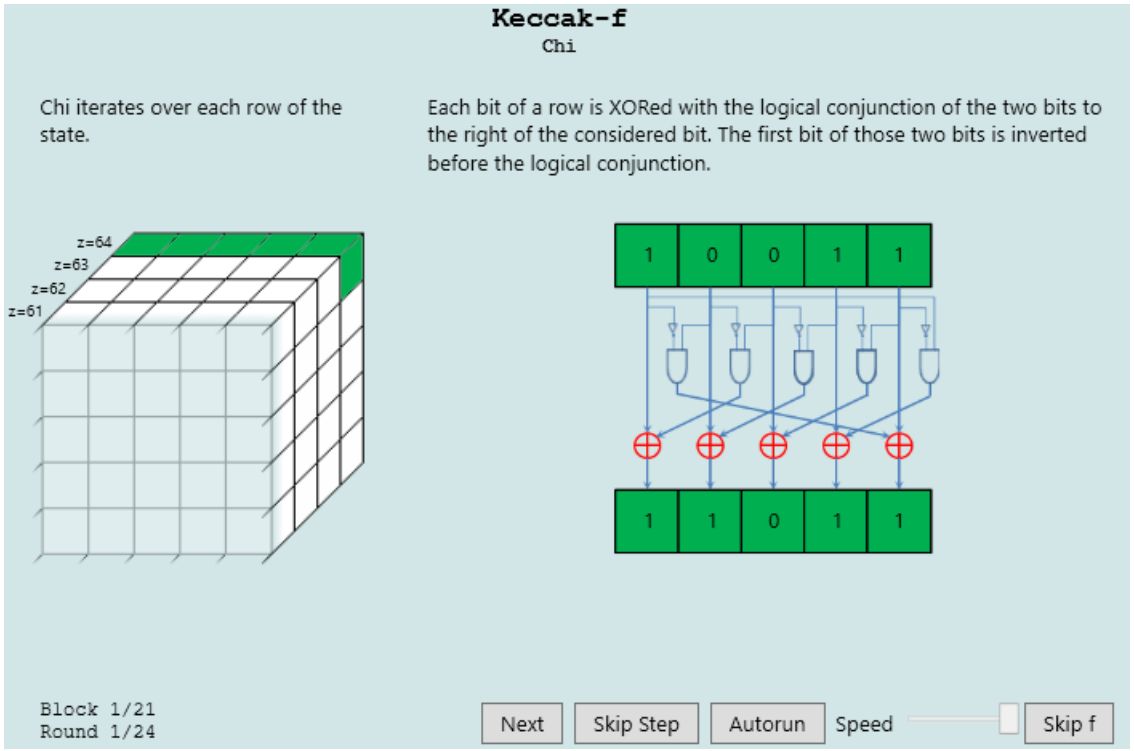


Рис. 15 – Chi операция

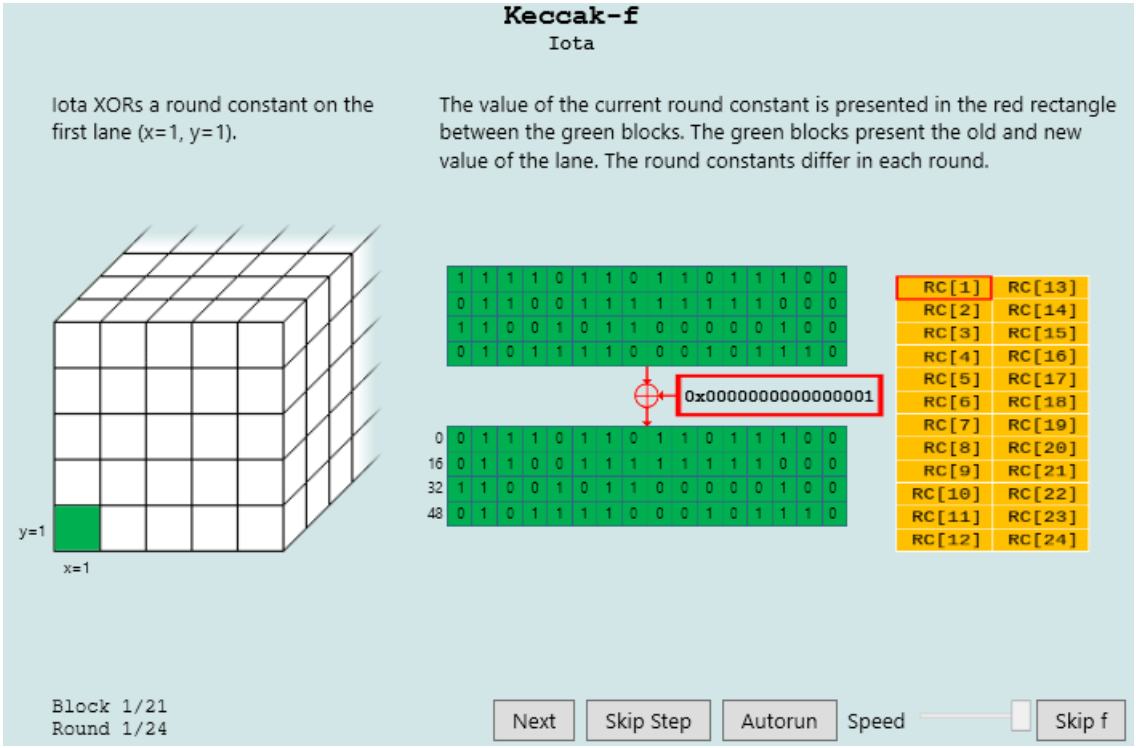


Рис. 16 – Iota операция

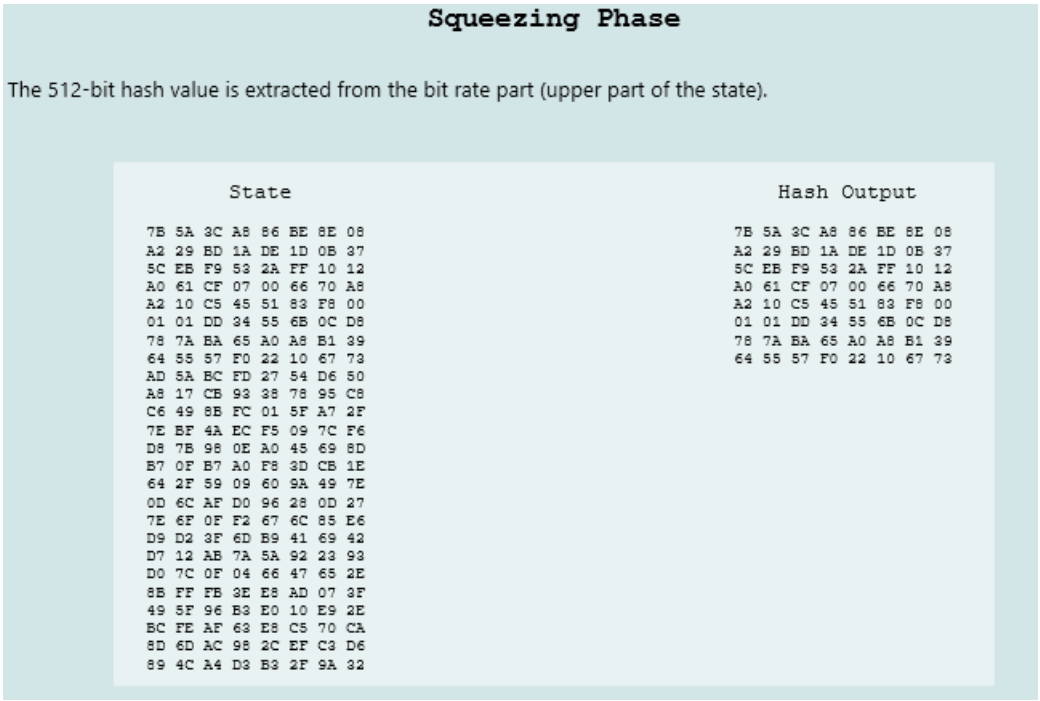


Рис. 17 – Завершающая фаза

В результате преобразований был получен хеш исходного текста:

```
7B 5A 3C A8 86 BE 8E 08 A2 29 BD 1A DE 1D 0B 37 5C EB F9 53 2A
FF 10 12 A0 61 CF 07 00 66 70 A8 A2 10 C5 45 51 83 F8 00 01 01
DD 34 55 6B 0C D8 78 7A BA 65 A0 A8 B1 39 64 55 57 F0 22 10 67
73
```

## 3. Средство оценивания лавинного эффекта

Для оценивания лавинного эффекта была написана программа на языке C++, которая считывает хеши исходного и измененного текстов из файлов и сравнивает их содержимое.

```
#include <bits/stdc++.h>

using namespace std;

void solve(ifstream &source, ifstream& changed) {

    char c1;
    char c2;
    int count = 0;

    while (!source.eof() && !changed.eof()) {
        source.read(&c1, sizeof(c1));
        changed.read(&c2, sizeof(c2));

        for (int i = 0; i < sizeof(c1) * 8; i++) {
            if ((c1 & 1) != (c2 & 1)) count++;
            c1 >>= 1;
            c2 >>= 1;
        }
    }

    if (source.eof() != changed.eof()) {
        cout << "invalid size of files";
        return;
    }

    cout << "Different bits count: " << count;
}

int main() {
    ifstream source( "template.bin", std::ios::binary);
    ifstream changed( "out.bin", std::ios::binary);

    if (!source.is_open() || !changed.is_open()) {
        cout << "cannot open files";
        return 1;
    }

    solve(source, changed);

    source.close();
    changed.close();
    return 0;
}
```

## 4. Оценивание лавинного эффекта

Оценим лавинный эффект для алгоритма SHA3-512 с использованием CrypTool 2 и описанной выше программы.

Табл. 2 – Оценивание лавинного эффекта для SHA3-512

| **Функция** | **Изменение** | **Добавление** | **Удаление** | **Среднее** | **%** |
|---|---|---|---|---|---|
| SHA3-512 | 264 | 261 | 284 | 270 | 53 |

Видно, что SHA3-512, как и рассмотренные ранее алгоритмы хеширования, показываем близкий к 50 процент различий между файлами при изменении, добавлении или удалении одного символа, что является хорошим результатом.

## 5. Выводы

В ходе исследования хеш-функции SHA3 был рассмотрен ее алгоритм. Визуализирован с использованием CrypTool 2 процесс раундовых преобразований и показан результат этих преобразований в виде сгенерированного хеша. Проведена оценка лавинного эффекта и определено, что процент измененных бит в сравнении с хешем исходного текста составляет примерно 53%.

# КОНТРОЛЬ ЦЕЛОСТНОСТИ ПО КОДУ HMAC

## 1. Задание

1. Выбрать текст на английском языке (не менее 1000 знаков), добавить собственное ФИО и сохранить в файле формата .TXT.

2. Придумать пароль и сгенерировать секретный ключ утилитой *Indiv.Procedures->Hash-> Key Generation* из Cryptool 1. Сохранить ключ в файле формата .TXT. Прочитать Help к этой утилите.

3. Сгенерировать HMAC для имеющегося текста и ключа с помощью утилиты *Indiv.Procedures->Hash-> Generation of HMACs*. Сохранить HMAC в файле формата .TXT. Прочитать Help к этой утилите.

4. Передать пароль, HMAC (и его характеристики), исходный текст и модифицированный текст коллеге, не раскрывая, какой текст является корректным. Попросите коллегу определить это самостоятельно.

## 2. Параметры ключа

Пароль: *mironchik8382*

Хеш-функция генерации ключа: MD5

Соль: 265006334

Количество итераций: 1000

Длина ключа: 16 байт

## 3. Параметры HMAC

HMAC хеш-функция: SHA-256

HMAC режим: ключ перед сообщением

Сгенерированный хеш:

```
17 5E 33 A6 C2 8D 84 F3 31 7E 17 24 84 C9 D5 7E 5E 51 8B D3 1C
EC 58 F1 54 B0 27 5A 6D FA C9 C3
```

## 4. Действия передающей стороны

1. Генерация ключа на основе известной информации о пароле и других данных, необходимых для генерации ключа.

2. Генерация хеша для текста на основе известных значений ключа и параметров HMAC.

3. Отправка текста и его хеша принимающей стороне. В нашем случае принимающей стороне необходимо также передать данные для генерации ключа и параметры HMAC.

**5. Действия принимающей стороны**

1. Генерация ключа и хеша для текста аналогично п.1 и п.2 из действий передающей стороны

2. Сверка сгенерированного хеша и хеша, полученного от передающей стороны. Если хеши совпали, значит текст соответствует отправленному передающей стороной и не был изменен.

**6. Выводы**

В ходе изучения контроля целостности по коду HMAC было проведено практическое исследование возможности определить подлинность текста – его соответствие отправленному передающей стороной. На основе ключа и согласованных с принимающей стороной параметров HMAC был сгенерирован хеш для исходного текста. Принимающая сторона получила хеш, исходный и измененный тексты, и, сгенерировав хеши для каждого из полученных текстов, определила подлинный, не подвергавшийся изменениям.

# АТАКА ДОПОЛНИТЕЛЬНОЙ КОЛЛИЗИИ НА ХЕШ-ФУНКЦИЮ

## 1. Задание

1. Сформировать два текста на английском языке - один истинный, а другой фальсифицированный. Сохранить тексты в файлах формата *.txt

2. Утилитой *Analysis-> Attack on the hash value...* произвести модификацию сообщений для получения одинакового дайджеста. В качестве метода модификации выбрать *Attach characters-> Printable characters.*

3. Проверить, что дайджесты сообщений действительно совпадают с заданной точностью.

4. Сохранить исходные тексты, итоговые тексты и статистику атаки для отчета.

5. Зафиксировать временную сложность атаки для 8, 16, 32,40, 48, … бит совпадающих частей дайджестов.

## 2. Описание атаки

Принцип атаки заключается в том, что при сравнительно малой длине хеша с большой вероятностью возможны коллизии при увеличении количества просмотренных вариантов исходного текста.

Например, имеются 2 аудитории, в которых находятся студенты. У каждого студента есть день рождения, от 1 до 365. Если посадить в каждую из аудиторий по 16 человек, то с вероятностью примерно в 0.5 найдется одна пара студентов, имеющих одинаковый день рождения.

В случае с реальными текстами ситуация выглядит точно так же, отличается лишь количество возможных вариантов “дня рождения”.

## 3. Атака

Проведем атаку на исходный текст и текст с добавленной в конец строкой *06.07.2021*.

Для этого найдем воспользуемся утилитой программы CrypTool 1 по поиску коллизий. Параметры, использованные для проведения атаки, указаны на рис. 18.

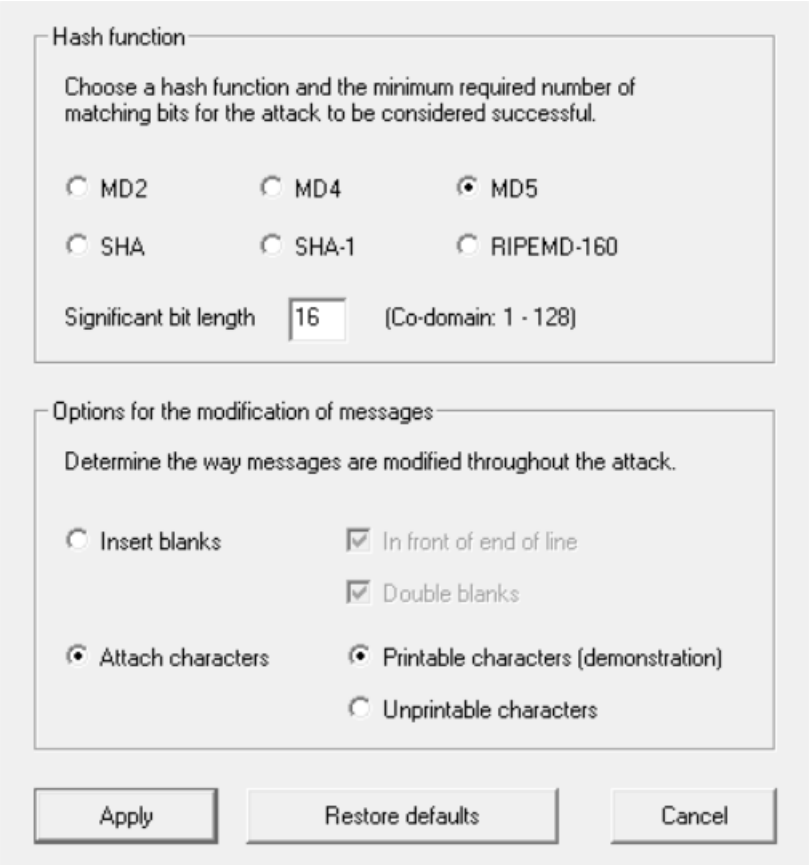


Рис. 18 – Параметры атаки

В результате в исходное и модифицированное сообщение было добавлено по 10 байт дополнительного текста, что позволило получить одинаковые первые 16 бит хеша:

```
B8 93 9F CE CF FE 26 61 9E F1 5A 3B 0B 3B 15 4A
B8 93 82 C6 12 24 61 55 D3 BE BC 97 95 36 6A 6A
```

Заметно, что хеши для исходного и модифицированного текста отличаются, однако совпадают первые 16 бит.

Проведем аналогичную атаку с использованием хеша MD5 для большего количества требуемых совпадающих бит.

Табл. 3 – Результаты атаки дополнительной коллизии

| **Количество совпадающих бит** | **Оценка временных затрат** |
|---|---|
| 16 | 0s |

| | |
|---|---|
| 24 | 0.07s |
| 32 | 2.77s |
| 40 | 26s |
| 48 | 4m |
| 56 | 1h 12m |
| 64 | 19h |
| 72 | 13d |
| 80 | 205d |
| 88 | 9y |
| 96 | 140y |
| 104 | 2.3e3y |
| 112 | 3.7e4y |
| 120 | 5.9e5y |
| 128 | >1.4e94y |

## 4. Выводы

Была исследована атака дополнительной коллизии на хеш функцию. Рассмотрен ее принцип действия на примере парадокса “дня рождения”, а также проведена атака с использованием средств CrypTool 1 для 16 совпадающих бит. Проведена оценка временных затрат на проведение атаки для 24-128 совпадающих бит, результаты которой представлены в табл. 3.