**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационной безопасности**

отчет

**по лабораторной работе №6**

**по дисциплине «Криптография и защита информации»**

Тема: Изучение хэш-функций

|  |  |  |
| --- | --- | --- |
| Студент гр. 9381 |  | Колованов Р.А. |
| Преподаватель |  | Племянников А.К. |

Санкт-Петербург

2022

**Цель работы.**

Исследовать хэш-функции MD5, SHA-256, SHA-512, SHA-3, код контроля целостности HMAC и проанализировать атаки дополнительной коллизии на хэш-функцию. Получить практические навыки работы с хэш-функциями и атакой на них, в том числе и в программном продукте Cryptool 1 и 2.

**Основные теоретические положения.**

***Хэш-функции.***

Хэш-функцией (*hash function*) называется математическая или иная функция, которая для строки произвольной длины вычисляет некоторое целое значение или некоторую другую строку фиксированной длины.

Хэш-значение может также называться дайджестом (*digest*) или отпечатком (*fingerprint*) сообщения.

Алгоритмы хэширования также называют бесключевыми дайджестами сообщений (*nonkeyed message digest*).

Однонаправленная функция *H(M)* применяется к сообщению длины *M* и возвращает значение фиксированной длины *h* (ℎ = 𝐻(𝑀), где *h* имеет длину *m*).

Однонаправленные функции имеют дополнительные свойства, позволяющие отличать их от обычных функций, которые вычисляют значение фиксированной длины по входным данным:

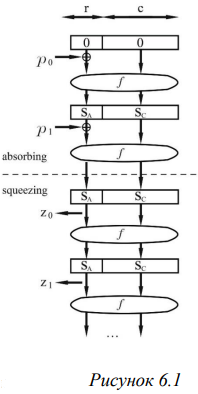
1. Зная *M*, легко вычислить *h*;
2. Зная *H*, трудно определить *M*, для которого 𝐻(𝑀) = ℎ;
3. Зная *M*, трудно определить другое сообщение, *M1*, для которого 𝐻(𝑀) = 𝐻(𝑀*1*).

Понятие идеальной хэш-функции:

* Для нового сообщения произвольной длины генерируется хэш фиксированной длины, который представляет собой случайную строку нулей и единиц. Эта пара (сообщение, хэш) образует запись в таблице;
* Для сообщения, которому ранее уже был сгенерирован хэш, дайджест выдается из соответствующей записи в таблице;
* Хэш для нового сообщения выбирается независимо от предыдущих значений хэша;
* Отношения между возможными сообщениями и возможными хэшами – «многие к одному».

***Хэш-функция SHA-3.***

В основе Keccak (SHA-3) лежит конструкция под названием Sponge – губка. Сам алгоритм состоит из 2-х этапов:

1. Впитывание – *Absording*. На каждом шаге очередной блок сообщения *pi* длиной *r* подмешивается к части внутреннего состояния *S*, которая затем целиком модифицируется функцией *f*-многораундовой бесключевой псевдослучайной перестановкой;
2. Отжатие – *Squeezing*. Чтобы получить хэш, функция *f* многократно применяется к состоянию, и на каждом шаге сохраняется кусок размера *r* до тех пор, пока не получим выход *Z* необходимой длины (путем конкатенации).

Обобщенная схема работы алгоритма представлена на рисунке 6.1.

***Контроль целостности по коду HMAC.***

HMAC – один из механизмов проверки целостности информации, позволяющий гарантировать то, что данные, передаваемые или хранящиеся в ненадёжной среде, не были изменены посторонними лицами. Механизм HMAC использует MAC — стандарт, описывающий способ обмена данными и способ проверки целостности передаваемых данных с использованием секретного ключа. Два клиента, использующие HMAC, как правило, разделяют общий секретный ключ. HMAC — надстройка над MAC.

Алгоритм HMAC можно записать в виде одной формулы:



где ⊕ – операция XOR, ∥ – конкатенация, *К* – секретный ключ, *ipad* – блок вида (0x36 0x36 0x36 ... 0x36), где байт 0x36 повторяется *b* раз, *H* – хэш-функция, *opad* – блок вида (0x5C 0x5C 0x5C ... 0x5C), где байт 0x5c повторяется *b* раз.

***Атака дополнительной коллизии на хэш-функцию.***

Атака дополнительной коллизии основана на одной из проблем парадокса дня рождений. Он заключается в следующем: в группе, состоящей из  или более человек, вероятность совпадения дней рождения (число и месяц) хотя бы у двух людей превышает 50%. Применительно к хэш-функции это означает, что сложность атаки, целью которой является поиск двух сообщений с одинаковыми значением хэш-функции, пропорционально , где N – длина хэш-кода.

**Ход работы.**

***Исследование лавинного эффекта MD5, SHA-1, SHA-256, SHA512.***

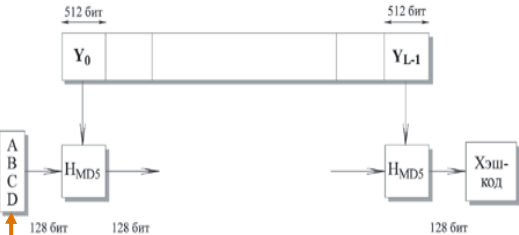
*Задание.*

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:
   1. Измените (добавлением, заменой, удалением символа) исходный файле;
   2. Зафиксировать количество измененных битов в дайджесте модифицированного сообщения;
   3. 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.

*Основные параметры и обобщенная схема хэш-функции MD5.*

Основные параметры хэш-функции MD5:

* Длина сообщения кратна 512 битам, в последний блок добавляются недостающие биты (в виде 1000…000) и 64-битное представление длины исходного сообщения ;
* Длина хэша – 128 бит;
* Число раундов – 4, в каждом раунде 16 итераций;
* MD-буфер – 4 регистра (A, B, C, D) по 32 бита, в сумме 128 бит.

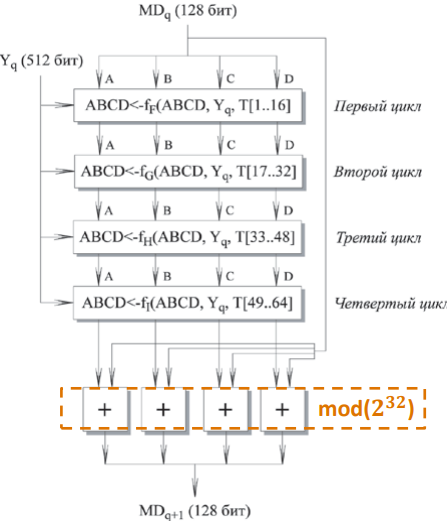




Функция сжатия :

* Каждый цикл переопределяется значение буфера ABCD;
* Т – массив вычисляемых величин (по 32 бита).





Цикл сжатия :

* Каждый цикл состоит из 6 шагов;
* *f* – одна из элементарных функций *fF*, *fG*, *fH* или *fI*:

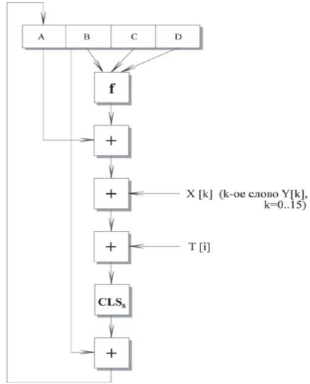








* Сложение выполняется по модулю ;
* 𝐶𝐿𝑆5 циклический сдвиг влево на 5 разрядов.



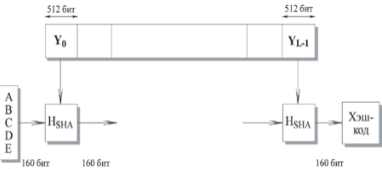
Свойства MD5:

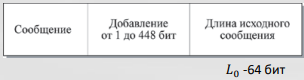
* Каждый бит хэш-кода является функцией от каждого бита входа;
* Комплексное повторение элементарных функций обеспечивает хорошее перемешивание результата;
* MD5 является наиболее сильной хэш-функцией для 128-битного хэш-кода.

*Основные параметры и обобщенная схема хэш-функции SHA-1.*

Основные параметры хэш-функции SHA-1:

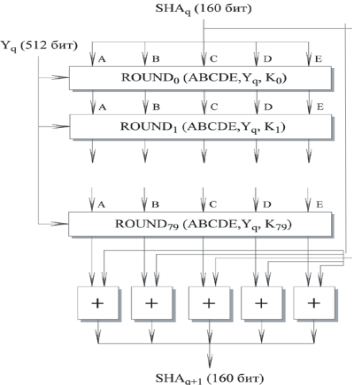
* Длина сообщения кратна 512 битам, в последний блок добавляются недостающие биты (в виде 1000…000) и 64-битное представление длины исходного сообщения ;
* Длина хэша – 160 бит;
* Число раундов – 80;
* MD-буфер – 5 регистров (A, B, C, D, E) по 32 бита, в сумме 160 бит.



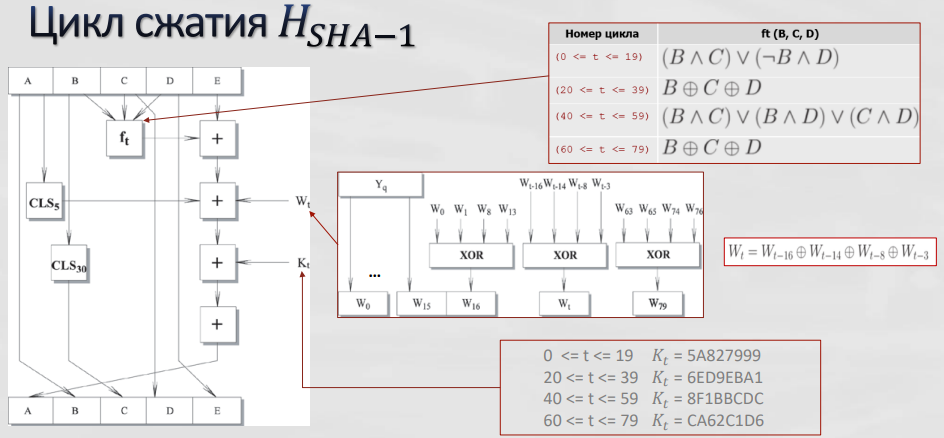


Функция сжатия :

* Состоит из 80 циклов обработок, имеющих одинаковую структуру;
* Каждый цикл переопределяет 160-битное значение буфера ABCDE;
* В каждом цикле используется дополнительная константа 𝐾t принимающая 4 различных значения;
* Сложение выполняется по модулю 232.

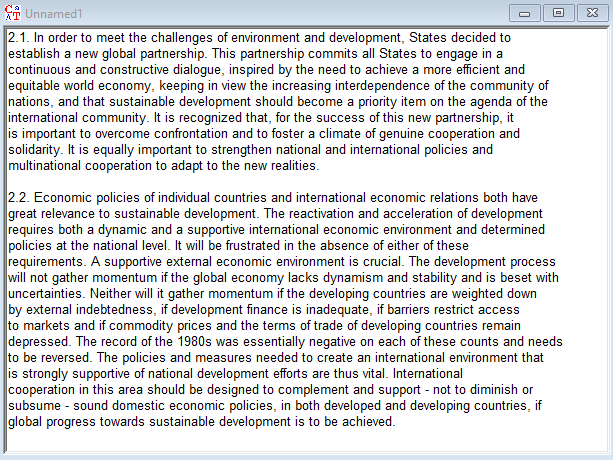


Цикл сжатия :

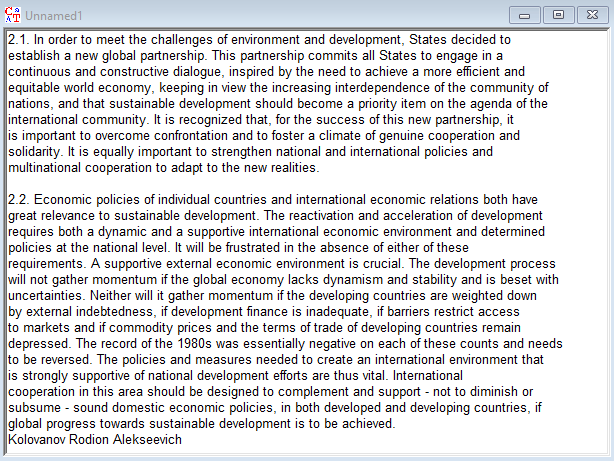


*Исследование лавинного эффекта хэш-функций MD5, SHA-1, SHA-256, SHA-512.*

Был выбран следующий открытый текст, размер которого составляет не менее 1000 символов:



В конец выбранного открытого текста была добавлена строка «Kolovanov Rodion Alekseevich»:



Далее была открыта утилита «Indiv. Procedures -> Hash -> HashDemonstration», в которой поочередно выбирались хэш-функции MD5, SHA-1, SHA-256, SHA-512.

Для каждой из рассматриваемых хэш-функций зафиксируем количество измененных бит в хэше при добавлении/изменении/удалении одного символа исходного сообщения, а также вычислим среднее количество измененных бит хэша:

|  |  |  |  |
| --- | --- | --- | --- |
| Хэш-функция | Изменение | Количество измененных бит хэша | Среднее количество измененных бит |
| MD5 | Добавление одного символа | 57.03% (73 из 128) | 53.39% (68.33 из 128) |
| Изменение одного символа | 50.78% (65 из 128) |
| Удаление одного символа | 52.34% (67 из 128) |
| SHA-1 | Добавление одного символа | 46.88% (75 из 160) | 50.63% (81 из 160) |
| Изменение одного символа | 50.00% (80 из 160) |
| Удаление одного символа | 55.00% (88 из 160) |
| SHA-256 | Добавление одного символа | 53.52% (137 из 256) | 50.91% (130.33 из 256) |
| Изменение одного символа | 52.34% (134 из 256) |
| Удаление одного символа | 46.88% (120 из 256) |
| SHA-512 | Добавление одного символа | 49.61% (254 из 512) | 49.54% (253.67 из 512) |
| Изменение одного символа | 47.66% (244 из 512) |
| Удаление одного символа | 51.37% (263 из 512) |

Как видно из результатов, при добавлении, изменении или удалении одного символа открытого текста среднее количество измененных бит хэша составляет около 51% от общего количества бит хэша.

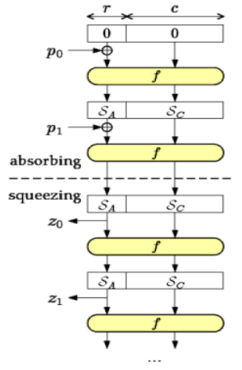
***Хэш-функция SHA-3.***

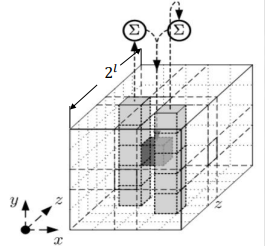
*Задание.*

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

*Основные параметры и обобщенная схема хэш-функции SHA-3.*

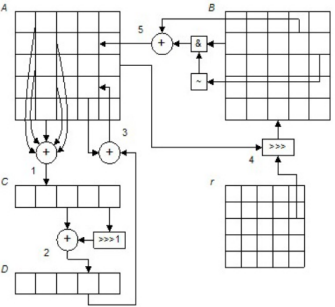
Хэш-функция Keccak:

* Является производительной в аппаратной реализации;
* Существует возможность реализации на миниатюрных встраиваемых устройствах;
* В основе Keccak лежит конструкция под названием Sponge (губка);
* Алгоритм состоит из двух этапов:
  + Absorbing (впитывание). На каждом шаге очередной блок сообщения pi длиной r подмешивается к части внутреннего состояния S, которая затем целиком модифицируется функцией f – многораундовой бесключевой псевдослучайной перестановкой
  + 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).

Внутреннее состояние:

* Текущее внутреннее состояние представлено в виде набора битов, сгруппированных в виде виртуального объекта в трёхмерном пространстве (трёхмерного массива);
* Объект можно разбить на плоскости или точнее слои вдоль трёх осей координат, а элементы каждого слоя — на фрагменты в виде столбцов или строк;
* Трёхмерное представление текущего внутреннего состояния помогает применить набор простейших логических операций (XOR, AND, NOT) оптимальным образом для реализации псевдослучайных перестановок.

Бесключевая псевдослучайная перестановка *f*:

* Раундовые ключи выполняют 5-ти шаговую обработку внутреннего состояния;
* Количество раундов равно 12 + 2\*l для состояния размера 25 \* 2l бит (0 <= l <= 6).

Оценка криптостойкости:

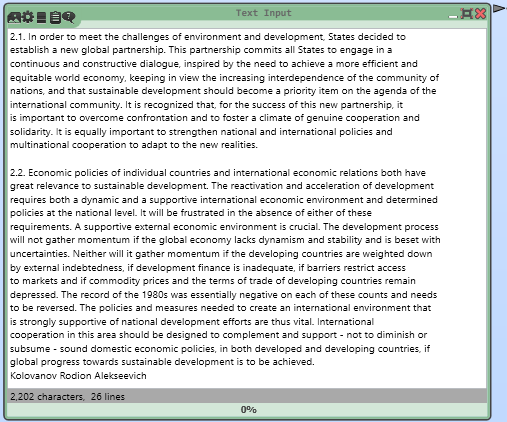
* Стойкость неотличима от стойкости идеальной хэш-функции с размером дайджеста, равным c/2, при условии, что f – идеальная функция перестановки;

Масштабируемость:

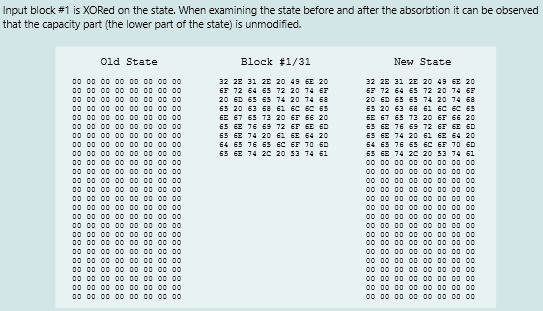
* Функцию перестановки f пользователь может выбирать самостоятельно из набора предопределенных функций;
* Каждая функция из набора обрабатывает внутреннее состояние определенного размера ;
* Для того, чтобы в реализации использовалась функция f, обрабатывающая состояние размера |S|, необходимо, чтобы r + c = |S|;
* Количество раундов n применения функции f вычисляется как n = 12 + 2\*l, где .

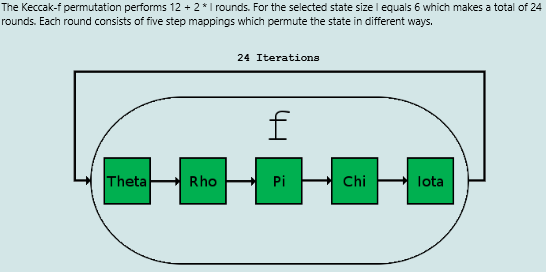
*Работа хэш-функции SHA-3.*

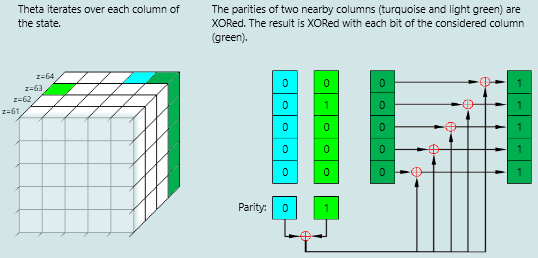
Для начала был открыт шаблон Keccak Hash (SHA-3) в CrypTool 2 (с настройками Adjust manually = ON, Keccak version = SHA3-512). В качестве исходного текста был взят исходный текст из предыдущего пункта:

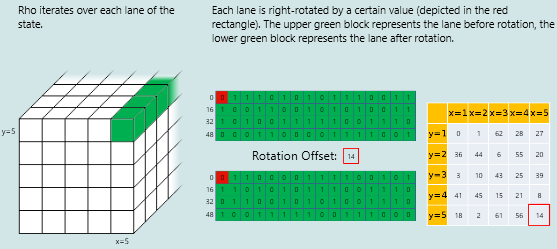


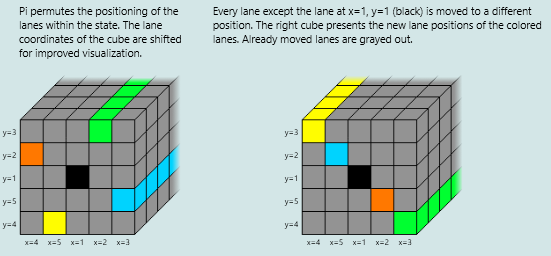
Скриншоты работы первого раунда:

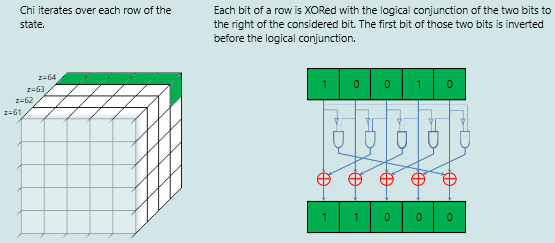


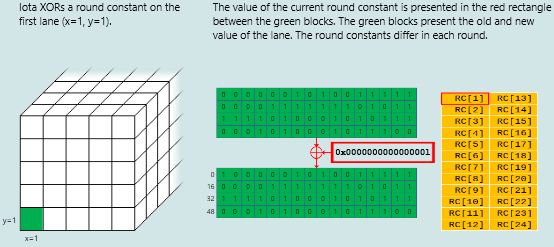




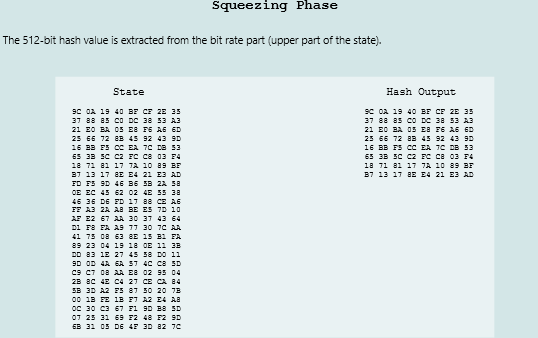








Скриншот заключительного этапа:



Полученное значение дайджеста:

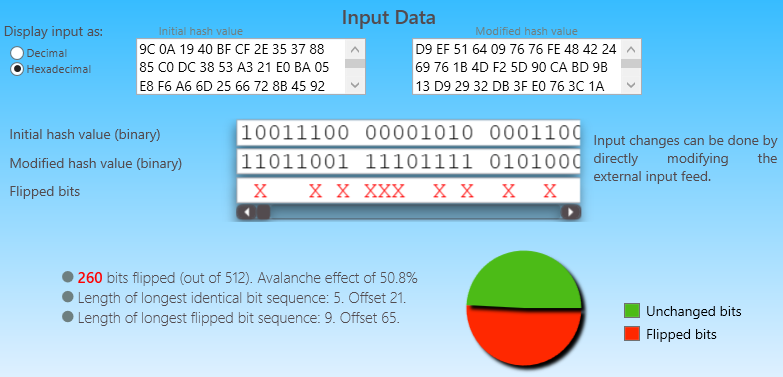
9C 0A 19 40 BF CF 2E 35 37 88 85 C0 DC 38 53 A3 21 E0 BA 05 E8 F6 A6 6D 25 66 72 8B 45 92 43 9D 16 BB F5 CC EA 7C DB 53 65 3B 5C C2 FC C8 03 F4 18 71 81 17 7A 10 89 BF B7 13 17 8E E4 21 E3 AD

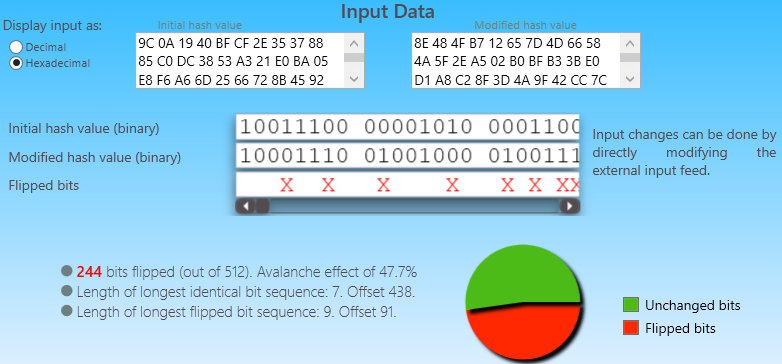
*Вычисление лавинного эффекта для хэш-функции SHA-3.*

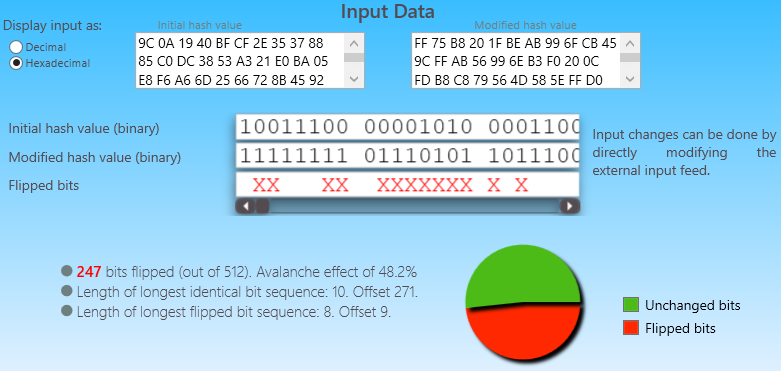
Далее для исходного текста были выполнены те же модификации, которые использовались в предыдущих пунктах для вычисления лавинного эффекта хэш-функций MD5, SHA-1, SHA-256, SHA-512. Для модифицированных текстов были вычислены дайджесты:

|  |  |
| --- | --- |
| Изменение | Хэш SHA-3 |
| Без изменений | 9C 0A 19 40 BF CF 2E 35 37 88 85 C0 DC 38 53 A3 21 E0 BA 05 E8 F6 A6 6D 25 66 72 8B 45 92 43 9D 16 BB F5 CC EA 7C DB 53 65 3B 5C C2 FC C8 03 F4 18 71 81 17 7A 10 89 BF B7 13 17 8E E4 21 E3 AD |
| Добавление одного символа | FF 75 B8 20 1F BE AB 99 6F CB 45 9C FF AB 56 99 6E B3 F0 20 0C FD B8 C8 79 56 4D 58 5E FF D0 8F C5 19 F5 B7 BE 54 45 B9 95 D1 BA 69 9C 85 34 41 BC 81 46 CD 2B 2D C8 0A F5 BF 64 5B 0B 2E 59 2E |
| Изменение одного символа | 8E 48 4F B7 12 65 7D 4D 66 58 4A 5F 2E A5 02 B0 BF B3 3B E0 D1 A8 C2 8F 3D 4A 9F 42 CC 7C C5 7E 59 3C 06 7F FF 0B E7 2F E0 5B 2E 63 B8 D9 12 35 D1 3A 48 0E C6 44 6D B9 95 BB C1 B7 CF A0 C4 EF |
| Удаление одного символа | D9 EF 51 64 09 76 76 FE 48 42 24 69 76 1B 4D F2 5D 90 CA BD 9B 13 D9 29 32 DB 3F E0 76 3C 1A 26 26 7A 97 40 47 2F 71 9C 3D 11 13 36 77 F5 DA 52 0F 38 F9 9B A9 55 0E 43 C6 70 54 BF 90 B4 25 91 |

Для полученных дайджестов было найдено количество измененных бит в хэше при добавлении/изменении/удалении одного символа исходного сообщения, а также вычислено среднее количество измененных бит хэша при помощи шаблона «Avalanche (Hash)»:







|  |  |  |  |
| --- | --- | --- | --- |
| Хэш-функция | Изменение | Количество измененных бит хэша | Среднее количество измененных бит |
| SHA-3 | Добавление одного символа | 48.2% (247 из 512) | 48.89% (250,33 из 512) |
| Изменение одного символа | 47.7% (244 из 512) |
| Удаление одного символа | 50.8% (260 из 512) |

Как видно из результатов, при добавлении, изменении или удалении одного символа открытого текста среднее количество измененных бит хэша составляет около 49% от общего количества бит хэша.

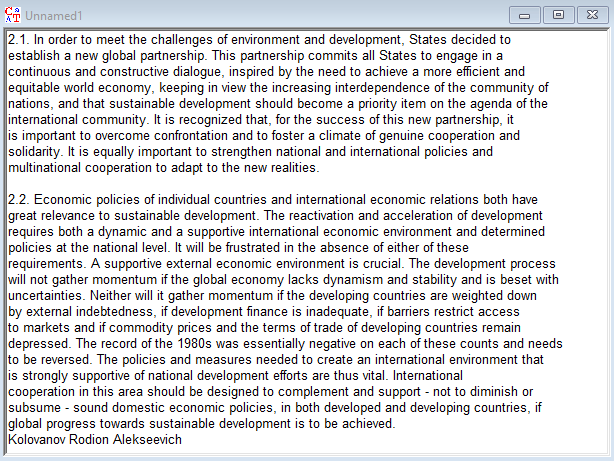
***Контроль целостности по коду HMAC.***

*Задание.*

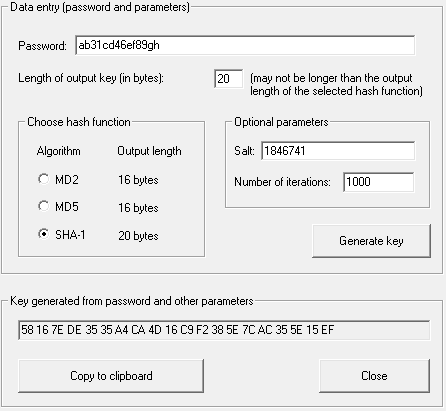
1. Выбрать текст на английском языке (не менее 1000 знаков), добавить собственное ФИО и сохранить в файле формата .TXT;
2. Придумать пароль и сгенерировать секретный ключ утилитой Indiv.Procedures -> Hash -> Key Generation из Cryptool 1. Сохранить ключ в файле формата .TXT. Прочитать Help к этой утилите;
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.

*Контроль целостности по коду HMAC.*

В качестве исходного текста был взят следующий текст, в конец которого была добавлена строка «Kolovanov Rodion Alekseevich»:

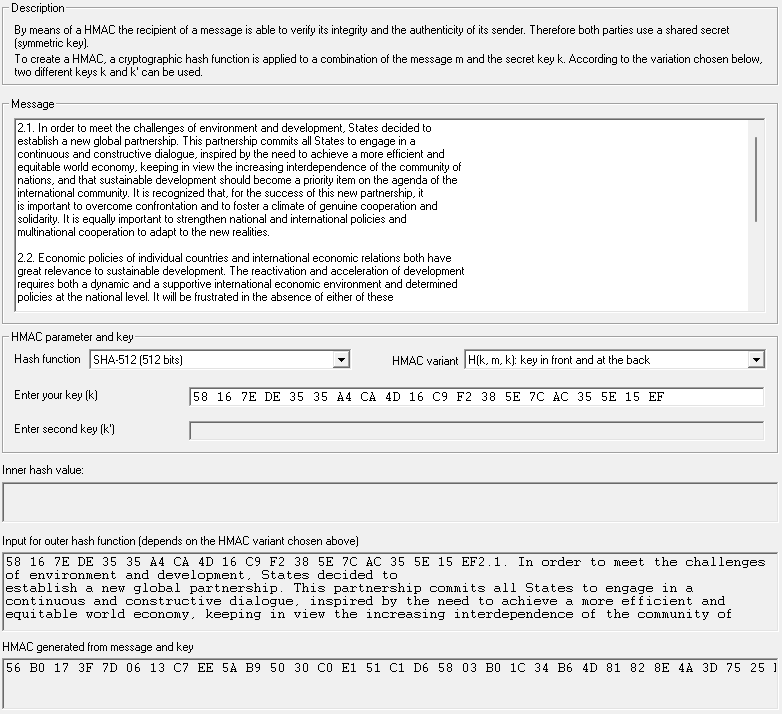


Далее при помощи утилиты Indiv.Procedures -> Hash -> Key Generation по паролю «ab31cd46ef89gh» был сгенерирован секретный ключ:



Сгенерированный секретный ключ: 58 16 7E DE 35 35 A4 CA 4D 16 C9 F2 38 5E 7C AC 35 5E 15 EF.

Далее при помощи утилиты Indiv.Procedures -> Hash -> Generation of HMACs для исходного текста и ключа был сгенерирован HMAC:



Сгенерированный HMAC: 56 B0 17 3F 7D 06 13 C7 EE 5A B9 50 30 C0 E1 51 C1 D6 58 03 B0 1C 34 B6 4D 81 82 8E 4A 3D 75 25 D1 46 30 75 92 EF EE 42 CF C0 A2 3D 1B B7 F5 8B 74 E7 2D 86 F2 B6 2B A1 4C 28 EB 49 87 24 90 CC.

От коллеги была получена следующая информация:

|  |  |
| --- | --- |
| Информация | Значение |
| Текст 1 | HMAC (Hash-based Message Authentication Code) is a type of a message authentication code (MAC) that is acquired by executing a cryptographic hash function on the data (that is) to be authenticated and a secret shared key. Like any of the MAC, it is used for both data integrity and authentication. Checking data integrity is necessary for the parties involved in communication. HTTPS, SFTP, FTPS, and other transfer protocols use HMAC. The cryptographic hash function may be MD-5, SHA-1, or SHA-256. Digital signatures are nearly similar to HMACs i.e they both employ a hash function and a shared key. The difference lies in the keys i.e HMACs use symmetric key(same copy) while Signatures use asymmetric (two different keys).  Processes and decisions pertinent to business are greatly dependent on integrity. If attackers tamper this data, it may affect the processes and business decisions. So while working online over the internet, care must be taken to ensure integrity or least know if the data is changed. That is when HMAC comes into use. Ptichkin Sergey Alekseevich. |
| Текст 2 | HMAC (Hash-based Message Authentication Code) is a type of a message authentication code (MAC) that is acquired by executing a cryptographic hash function on the data (that is) to be authenticated and a secret shared key. Like any of the MAC, it is used for both data integrity and authentication. Checking data integrity is necessary for the parties involved in communication. HTTP, FTP, and other transfer protocols use HMAC. The cryptographic hash function may be MD-11, SHA-3, or SHA-1024. Digital signatures are nearly similar to HMACs i.e they both employ a hash function and a shared key. The difference lies in the keys i.e HMACs use asymmetric key(same copy) while Signatures use symmetric (three different keys).  Processes and decisions pertinent to business are greatly dependent on integrity. If attackers tamper this data, it may affect the processes and business decisions. So while working online over the internet, care must be taken to ensure integrity or least know if the data is changed. That is when MAC comes into use. Ptichkin Sergey Alekseevich. |
| Пароль | mjf8q923jrjr8945 |
| Параметры генерации ключа | Соль: 265006334, количество итераций: 1000, хэш-функция: SHA-1. |
| HMAC | 15 F0 B5 02 86 B4 93 8D 8F 24 54 64 0A 2F 8D 37 0C 62 A7 1C DB DD C7 3B 21 DE 68 03 03 7B 88 E7 |
| Параметры генерации HMAC | Хэш-функция: SHA-256, тип H(m, k) |

Для обоих текстов по полученному паролю и характеристикам HMAC был вычислен HMAC.

HMAC для текста 1: 15 F0 B5 02 86 B4 93 8D 8F 24 54 64 0A 2F 8D 37 0C 62 A7 1C DB DD C7 3B 21 DE 68 03 03 7B 88 E7

HMAC для текста 2: AB BA 58 8E BA E2 B5 CF AA 11 8C 96 C0 4C 3E 98 78 00 F4 EC 02 61 10 6A 97 01 99 00 21 B0 0D 45

HMAC для текста 1 совпадает с полученным от коллеги HMAC, поэтому текст 1 является исходным, а текст 2 – модифицированным.

***Атака дополнительной коллизии на хэш-функцию.***

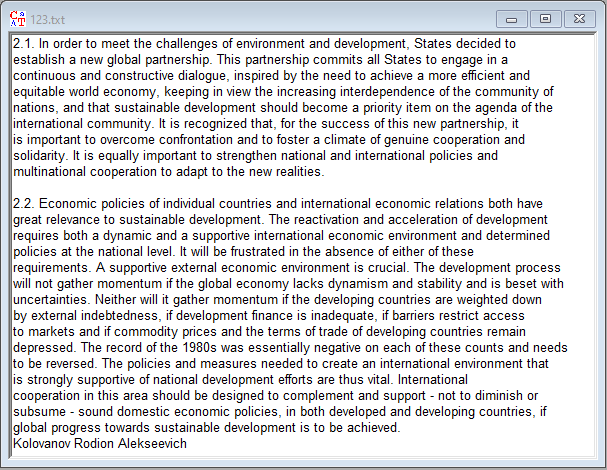
*Задание.*

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.

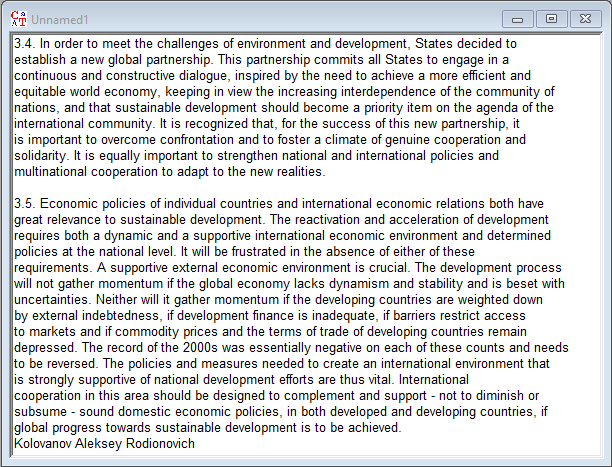
*Атака дополнительной коллизии на хэш-функцию.*

Для начала были сформированы два текста: один – истинный, другой – фальсифицированный.

Истинный текст:



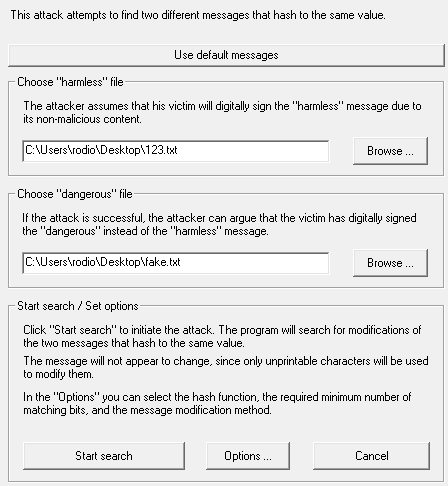
Фальсифицированный текст:

****

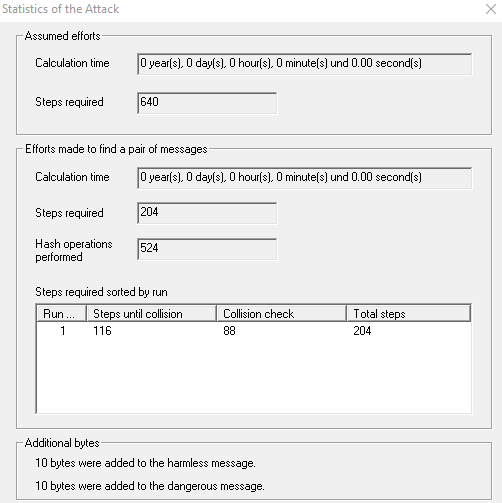
SHA-1 хэш для исходного сообщения: A7 A0 5B 84 02 AD 22 4B 70 C0 65 80 4F A5 F4 7B 5F 46 62 E4.

SHA-1 хэш для фальсифицированного сообщения: 86 BF 48 C3 90 6F 6B 65 F8 40 A8 D5 59 3D D6 03 68 05 01 C5.

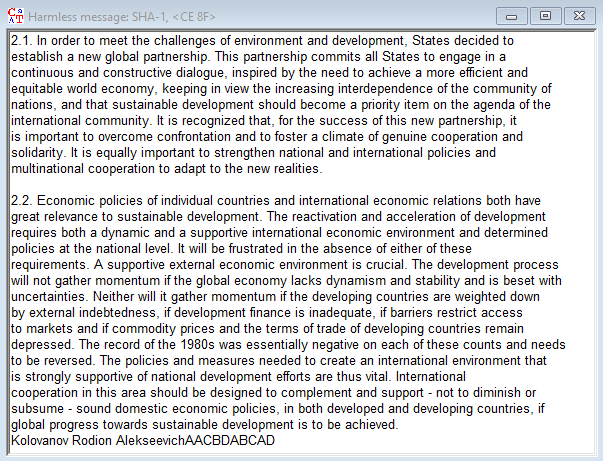
Далее при помощи утилиты Analysis -> Hash -> Attack on the hash value была произведена модификация сообщений для получения одинакового дайджеста:

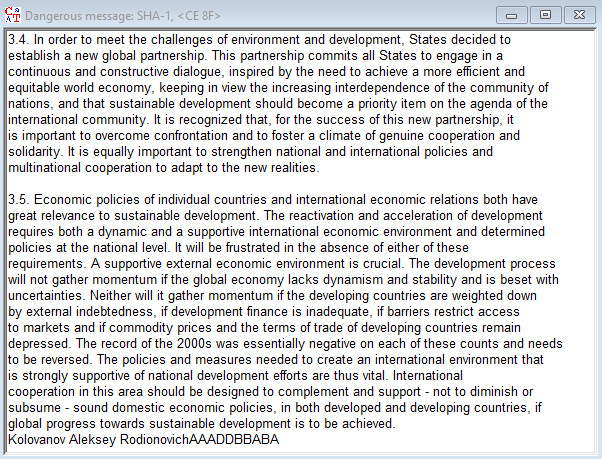
****

Статистика атаки:



Полученные итоговые тексты:





SHA-1 хэш для исходного сообщения после атаки: CE 8F B3 F1 AC 88 A9 9D D3 35 F8 0B 9E EE CB EB 04 B0 2A 05.

SHA-1 хэш для фальсифицированного сообщения после атаки: CE 8F FB 71 EF 7E 48 BC 50 AA F1 66 AC 45 0D E5 84 9B 94 C9.

Видно, что первые два байта (16 бит) хэша совпадают с заданной точностью (16 бит), отсюда атака прошла успешно.

Далее зафиксируем временную сложность атаки при различном количестве совпадающих частей хэша:

|  |  |
| --- | --- |
| Количество бит | Время атаки |
| 8 | 0 секунд |
| 16 | 0 секунд |
| 24 | 0,1 секунда |
| 32 | 1,72 секунды |
| 40 | 27,56 секунд |
| 48 | 3 минуты 7,21 секунд |
| 56 | ~ 1 час 30 минут |
| 64 | ~ 23 часа |
| 72 | ~ 15,9 дней |
| 80 | ~ 256 дней |
| 88 | ~ 11 лет |
| 96 | ~ 180 лет |
| 104 | ~ 2900 лет |
| 112 | ~ 46000 лет |
| 120 | ~ 7,5 \* 105 лет |

**Выводы.**

В ходе выполнения данной лабораторной работы были исследованы хэш-функции MD5, SHA-256, SHA-512, SHA-3 и код контроля целостности HMAC, а также была проанализирована атака дополнительной коллизии на хэш-функцию.

1. Лавинный эффект хэш-функций MD5, SHA-1, SHA-256, SHA-512:
   1. При помощи инструментария CrypTool 1 был изучен лавинный эффект для хэш-функций MD5, SHA-1, SHA-256, SHA-512. Было определено, что добавление, замена или удаление одного символа в исходном тексте приводит к изменению в среднем 51% битов дайджеста для всех хэш-функций. Отсюда рассмотренные хэш-функции обладают лавинным эффектом.
2. Хэш-функция SHA-3:
   1. При помощи демонстрационного примера была рассмотрена работа хэш-функции SHA-3. Было определено, что в основе SHA-3 лежит конструкция «Губка», алгоритм которой состоит из двух этапов: впитывание и отжатие. На этапе впитывания очередные блоки сообщения подмешиваются к части внутреннего состояния, а на этапе отжатия извлекаются очередные части дайджеста. На каждой итерации применяется многораундовая бесключевая псевдослучайная перестановка, которая может выбирается пользователем из набора предопределенных функций. В рассматриваемом SHA-3 используется функция f-1600, размер дайджеста составляет 512 бит, размер блока сообщения составляет 576 бит, размер внутреннего состояния составляет 1600 бит, а количество раундов равно 24.
   2. При помощи инструментария CrypTool 2 был изучен лавинный эффект для хэш-функции SHA-3. Было определено, что добавление, замена или удаление одного символа в исходном тексте приводит к изменению в среднем 49% битов дайджеста. Отсюда хэш-функция SHA-3 обладает лавинным эффектом.
3. Контроль целостности по коду HMAC:
   1. Был рассмотрен механизм для проверки целостности HMAC, позволяющий гарантировать то, что данные, передаваемые или хранящиеся в ненадёжной среде, не были изменены посторонними лицами. Вычисление кода HMAC осуществляется по формуле , где K – секретный ключ, opad и ipad – определенные константы, а text – исходный текст.
   2. При помощи известной информации о пароле, HMAC и методе генерации ключа и HMAC, была осуществлена проверка целостности двух сообщений, полученных от коллеги. В результате было выявлено, что одно из сообщений было модифицировано.
4. Атака дополнительной коллизии на хэш-функцию:
   1. Была проведена атака дополнительной коллизии на хэш-функцию SHA-1. Размер дайджеста для хэш-функции SHA-1 составляет 160 бит. Было определено, что получение коллизии части дайджеста размером до 48 бит занимает относительно мало времени (около пары минут). Для получения коллизии части дайджеста размером свыше 48 бит временная сложность атаки значительно растет, и для получения коллизии части дайджеста размером 96 бит временная сложность атаки составляет около 180 лет. Это подтверждает, что атака дополнительной коллизии имеет экспоненциальную сложность.

Были получены практические навыки работы с рассматриваемыми хэш-функциями и атакой на них с использованием приложения CrypTool 1 и 2.