**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ**

**«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)**

**Кафедра САПР**

**КУРСОВАЯ РАБОТА**

# по дисциплине «Алгоритмы и структуры данных» Тема: Исследование алгоритмов работы хеш функций

|  |  |  |
| --- | --- | --- |
| Студент гр. 3352 |  | Шушков В.А. |
| Преподаватель |  | Пестерев Д. О. |

Санкт-Петербург 2024

**ЗАДАНИЕ**

**НА КУРСОВУЮ РАБОТУ**

|  |  |
| --- | --- |
| Студент Шушков В.А. |  |
| Группа 3352 |  |
| Тема работы: Исследование алгоритмов работы хеш функций | |
| Исходные данные:  Python 3.12. PyCharm Community Edition 2024.1.6. | |
| Содержание пояснительной записки:  «Содержание», «Введение», «Основы хеширования», «Хеш-функция MD5», «Уязвимости MD5», «Программная реализация», «Результаты замеров» , «Код программы», «Заключение», «Список использованных источников» | |
| Предполагаемый объем пояснительной записки: Не менее 18 страниц. | |
| Дата выдачи задания: 19.11.2024 |  |
| Дата сдачи реферата: .12.2024 |  |
| Дата защиты реферата: .12.2024 |  |
| Студент | Шушков В.А. |
| Преподаватель | Пестерев Д.О. |

**АННОТАЦИЯ**

В данной работе рассматриваются алгоритмы работы хеш-функций и методы взлома паролей, когда злоумышленник имеет доступ только к хэшу. Обсуждаются три основных метода атаки. Для иллюстрации работы хеш-функций и методов их взлома была разработана программа на языке программирования Python. Проведены тесты, позволяющие оценить эффективность различных атак в зависимости от сложности паролей.

**SUMMARY**

In this paper, the algorithms of hash functions and methods for cracking passwords are examined when an attacker has access only to the hash. Three main attack methods are discussed. A program was developed in Python to illustrate the operation of hash functions and methods for their compromise. Tests were conducted to evaluate the effectiveness of various attacks based on the complexity of passwords.

**СОДЕРЖАНИЕ**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Глава 1. Основы хеширования | 6 |
| 1.1. | Понятие хеш-функции | 6 |
| 1.2. | Принципы работы хеш-функции | 6 |
| 2. | Глава 2. Хеш-функция MD5 | 7 |
| 2.1 | Алгоритм работы MD5 | 7 |
| 3. | Глава 3. Уязвимости MD5 | 8 |
| 3.1 | Обзор известных атак на MD5 | 8 |
| 3.2. | Примеры успешных атак на MD5 | 10 |
| 4. | Глава 4. Программная реализация | 11 |
| 5. | Глава 5. Результаты замеров. | 11 |
| 6. | Глава 6. Код программы. | 15 |
| 7. | Глава 7. Заключение | 17 |
| 8. | Глава 8. Список используемых источников | 18 |

**ВВЕДЕНИЕ**

Хеширование является важным аспектом информационной безопасности и обработки данных, обеспечивая целостность и защиту информации. Хеш-функции, такие как MD5, играют ключевую роль в этих процессах, преобразуя входные данные произвольного размера в фиксированные хеш-значения. Однако, несмотря на свою популярность, MD5 подверглась многочисленным атакам и уязвимостям, что поставило под сомнение её надежность в современных системах безопасности. В данной работе рассматриваются основные принципы хеширования, алгоритм MD5 и его уязвимости.

# Глава 1. Основы хеширования

* 1. **Понятие хеш-функции**

Хеш-функция — это алгоритм, который принимает входные данные произвольного размера и преобразует их в фиксированное значение фиксированной длины, называемое хешем или хеш-значением. Хеш-функции широко используются в различных областях, включая криптографию, хранение данных и проверку целостности.

* 1. **Принципы работы хеш-функций.**

Хеш-функции работают на основе нескольких ключевых принципов, которые обеспечивают их функциональность и безопасность. Вот основные из них:

* Входные данные произвольного размера: Хеш-функция принимает входные данные любого размера (например, строку, файл или поток данных) и преобразует их в хеш-значение фиксированной длины.
* Детерминированность: Для одного и того же входного значения хеш-функция всегда будет генерировать одно и то же хеш-значение. Это свойство позволяет использовать хеши для проверки целостности данных.
* Односторонность: Хеш-функции должны быть односторонними, что означает, что, зная хеш-значение, невозможно (или крайне сложно) восстановить оригинальные входные данные. Это свойство делает хеши полезными для хранения паролей
* Быстрота вычисления: Хеш-функции должны обеспечивать быстрое вычисление хеш-значений, что позволяет эффективно обрабатывать большие объемы данных.
* Стойкость к атакам: Хорошие хеш-функции должны быть устойчивыми к различным атакам, включая атаки по словарю и брутфорс, что делает их надежными для использования в системах безопасности.

# Глава 2. Хеш-функция MD5

* 1. **Алгоритм работы MD5**

MD5 (Message Digest Algorithm 5) — это криптографическая хэш-функция, разработанная Рональдом Ривестом в 1991 году. Она принимает сообщение произвольной длины на вход и генерирует на выходе хэш фиксированной длины (128 бит или 16 байт).

Несмотря на свою популярность в прошлом, MD5 больше не считается безопасной из-за обнаруженных уязвимостей, таких как коллизии (два разных входных сообщения могут давать одинаковый хэш). Тем не менее, алгоритм широко используется для проверки целостности данных.

1. **Подготовка сообщения**

* Сообщение сначала дополняется (padding), чтобы его длина соответствовала требованиям алгоритма : сообщение дополняется битом 1, а затем добавляется столько битов 0, чтобы его длина по модулю 512 стала равна 448.

К сообщению добавляются 64 бита, которые представляют собой исходную длину сообщения (в битах), записанную в виде младшего порядка (little-endian).

Таким образом, общее количество битов сообщения становится кратным 512 битам.

1. **Инициализация буфера**

* Алгоритм использует 4 32-битных регистра: A, B, C и D. Они инициализируются следующими значениями:

A = 0x67452301

B = 0xEFCDAB89

C = 0x98BADCFE

D = 0x10325476

Эти регистры будут обновляться на каждом этапе обработки данных.

1. **Основной цикл (обработка блоков)**

* Сообщение разбивается на блоки по 512 бит (64 байта). Каждый блок обрабатывается в цикле с использованием 4 раундов, каждый из которых состоит из 16 шагов.

Шаги раундов используют четыре нелинейные функции (F, G, H, I):

F (B, C, D) = (B AND C) OR ((NOT B) AND D)

G (B, C, D) = (B AND D) OR (C AND (NOT D))

H (B, C, D) = B XOR C XOR D

I (B, C, D) = C XOR (B OR (NOT D))

На каждом шаге:

Один из 16 32-битных слов блока сообщения используется как вход.

Применяется операция над регистром, функцией (F, G, H или I) и входным словом.

Применяется сдвиг и суммирование с использованием таблицы констант.

1. **Добавление результата блока**

* После обработки каждого блока 512 бит, обновлённые значения регистров A, B, C и D добавляются к их предыдущим значениям. Эти регистры будут итоговыми после обработки всех блоков.

1. **Финальный хэш**

* После обработки всех блоков результирующие регистры **A, B, C и D** объединяются в **128-битный хэш**. Регистр **A** идёт первым, затем **B**, **C** и **D**. Каждый из них представляется в виде 32-битного числа в **младшем порядке (little-endian)**. Результат — строка из 32 символов в шестнадцатеричном формате.

# Глава 3. Уязвимости MD5

* 1. **Обзор известных атак на MD5**

MD5, как криптографическая хэш-функция, на протяжении многих лет подвергалась многочисленным атакам, которые привели к утрате её криптографической надёжности. Ключевые уязвимости включают коллизии, атаки на предобразование и атаки на длину сообщения. Рассмотрим каждую из них подробнее:

1. **Коллизии**

* Коллизия возникает тогда, когда два разных входных сообщения дают одинаковый хэш. Для криптографической хэш-функции такая ситуация должна быть крайне редкой, однако в MD5 коллизии можно находить на практике из-за недостатков в алгоритме.
* **Первый практический пример** коллизий был продемонстрирован в 2004 году группой исследователей во главе с Ван Сяоинь. Им удалось сгенерировать две разные строки, которые имели одинаковый MD5-хэш.

**Типы коллизий**:

* Полная коллизия: два разных сообщения дают одинаковый хэш.
* Полуколлизия: два сообщения с одинаковым началом дают одинаковый хэш.

Коллизии MD5 делают алгоритм **непригодным для цифровых подписей** и других критически важных применений, где требуется гарантия уникальности хэша.

* **Пример Коллизии**

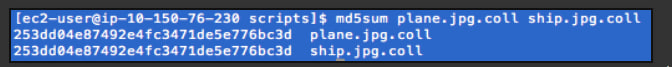
1. Инцидент с вредоносной программой Flame. В 2012 году злоумышленники использовали слабость в криптографическом алгоритме MD5 для создания неавторизованного цифрового сертификата Microsoft. Это позволило вредоносной программе маскироваться под легитимное обновление Microsoft, тем самым обманывая системы, заставляя их принять вредоносное ПО.
2. На Рис.1 можно увидеть , что у разных .jpg файлов одинаковый хэш , но при этом , на Рис.2 и Рис.3 , Мы можем заметить , что сами картинки разные.

Рис.1 – пример разных файлов с одинаковым хэшом.





Рис.2 – plane.jpg Рис.3 – ship.jpg

1. **Атаки на предобразование**

* Атака на предобразование — это метод, при котором злоумышленник пытается найти такое сообщение **M'**, хэш которого совпадает с хэшем известного сообщения **M**.
* Для MD5 атака на предобразование значительно упрощается из-за малой длины хэша (128 бит), что позволяет проводить **грубой перебор** (brute-force) для нахождения подходящего значения. Современные вычислительные мощности позволяют находить коллизии или предобразования намного быстрее, чем предполагалось при разработке алгоритма.

1. **Атаки на длину сообщения**

* MD5 уязвим к атакам на длину сообщения из-за своей внутренней структуры на основе циклического дополнения блоков. Атака на длину сообщения позволяет злоумышленнику добавить к уже известному хэшу дополнительную информацию и получить корректный хэш для модифицированного сообщения, не зная исходных данных.
  1. **Примеры успешных атак на MD5**

1. **Атака на пароли**

MD5 часто использовался для хранения хэшей паролей в базах данных. Однако из-за уязвимостей алгоритма такие хэши легко взламываются следующими методами:

1. **Перебор паролей (Brute-Force):**  
   Из-за короткой длины хэша и устаревшей конструкции, MD5 подвержен грубому перебору. Современные видеокарты и специализированное ПО (например, Hashcat) позволяют вычислять миллиарды хэшей MD5 в секунду, перебирая возможные пароли.
2. **Атака с использованием радужных таблиц (Rainbow Tables):**  
   Радужные таблицы — это заранее вычисленные хэши паролей для популярных комбинаций символов. Для MD5 такие таблицы уже существуют, что позволяет моментально подбирать простые пароли (например, "123456" или "password").
3. **Коллизии и предобразование:**  
   Злоумышленники могут использовать найденные коллизии или создавать хэши, соответствующие другим сообщениям, чтобы обойти проверки аутентификации.

* **Примеры атаки:**

1. В 2012 году была взломана база данных LinkedIn, где хэши паролей хранились с использованием MD5.
2. Злоумышленники смогли восстановить миллионы паролей, поскольку алгоритм MD5 не был защищён от атак перебором и коллизий.

# Глава 4. Программная реализация.

Основная функция запрашивает у пользователя ввод пароля, который затем хешируется с использованием MD5. После этого программа выполняет три метода атак на пароль: перебор, словарную атаку и радужную атаку, используя заранее подготовленный файл со списком паролей.

В методе перебора программа пытается найти соответствующий пароль, перебирая все числовые значения до заданного предела и сравнивая их хеши с целевым хешем. При этом выводится прогресс выполнения.

В словарной атаке программа загружает список паролей из файла и хеширует каждую строку, сохраняя результаты в словаре для быстрого поиска. После создания словаря программа ищет хеш, введенный пользователем, и выводит найденный пароль.

Радужная атака использует концепцию редукции, позволяя преобразовывать хеши в строки фиксированной длины. Программа создает набор редуцированных хешей и проверяет их на наличие совпадений с целевым хешем.

Кроме того, код включает функцию для тестирования времени выполнения хеширования для различных алгоритмов, что позволяет оценить производительность различных хеш-функций при обработке входных данных. Результаты тестирования визуализируются с помощью графиков.

В целом, этот код демонстрирует различные подходы к взлому хешированных паролей и позволяет оценить эффективность различных методов и алгоритмов хеширования.

# Глава 5. Результаты замеров.

Были проведены различные тесты с различными примерами паролей , которые отличались друг от друга.

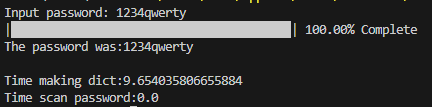


Рис.4 – Пример «простого» пароля для Словарной атаки

На данном примере мы использовали довольно популярный пароль , который состоит из близко находящихся символов , что создает большую уязвимость для взлома.

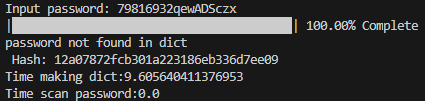


Рис.5 – пример «универсального» пароля для Словарной атаки

На данном примере можно заметить, что пароль состоит из разных универсальных символов. Данный пароль не состоит из популярных комбинаций, что способствует лучшей защите пароля.

Оба пароля были проверены на методе – «**Словарная атака**»

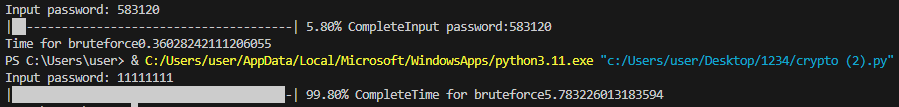


Рис.6 – пример паролей для атаки методом перебора

Можем заметить, что первый пароль, который состоит из меньшего кол-ва символов , взломался в несколько раз быстрее, чем пароль , который состоит из большего числа символов , несмотря на то , что длинный пароль не является «универсальным». Это показывает сильные и слабые стороны данного метода. Т.к. чем длиннее пароль, тем сложнее и дольше будет его взламывать программа , при этом , пароль никак не зависит от своей «универсальности» , ведь метод перебирает буквально все возможные вариации. Из вышесказанного можно сделать вывод , что метод перебора является неплохим методом атаки для короткого пароля .

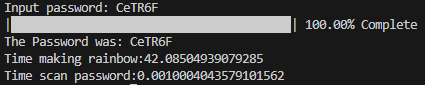


Рис.7 – пример использования Радужных таблиц.

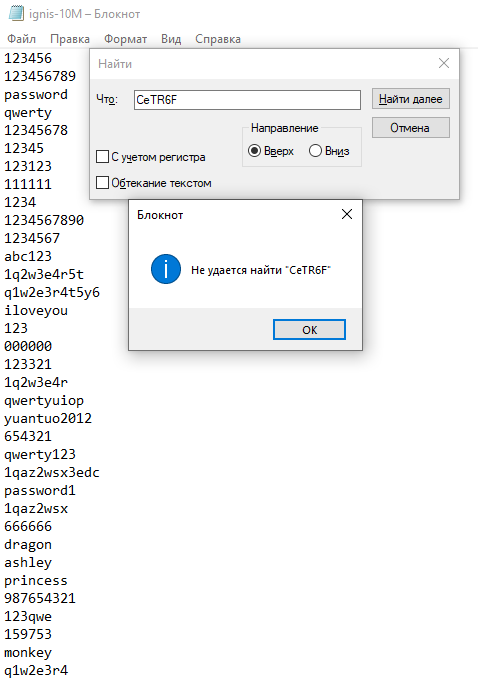


Рис.8 – Файл со всеми комбинациями паролей.

На изображении 7 дается на вход пример «Универсального» пароля, которого нету в списке доступных паролей , как показывает изображение 8. Благодаря своему методу Радужных таблиц , алгоритм переводит пароль в хэш , после чего прогоняет полученный хэш через функцию редукции , которая из полученного хэша генерирует некоторую строку заданной длинны ( в нашем случае длинна строки – 6 символов ).   
Данный метод позволяет получать доступ к различным универсальным паролям. Однако , как и в предыдущих методах , главной проблемой становится количество символов.

Далее мы рассмотрим время работы различных хеш-функций, таких как – Md5, SHA1, SHA256, SHA3\_256, Blake2b, Blake2s.

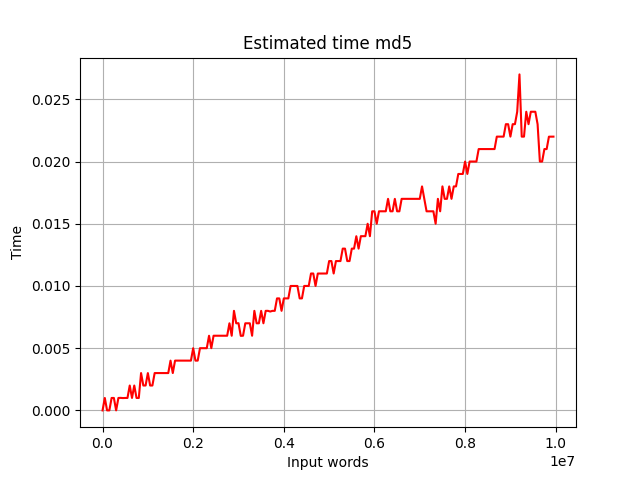
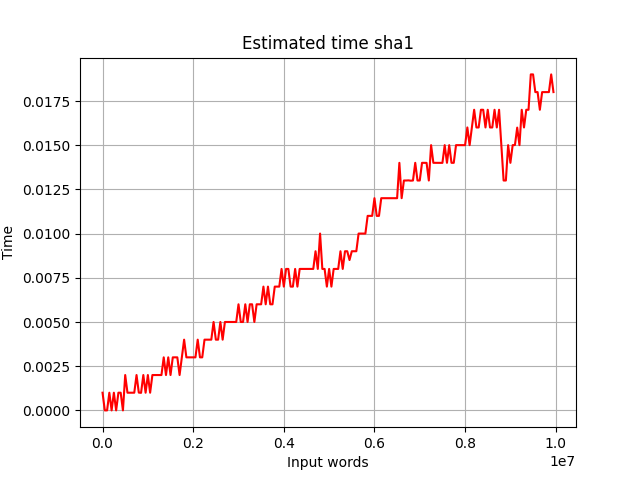
 

Рис.9 – график времени MD5 Рис.10 – график времени SHA1

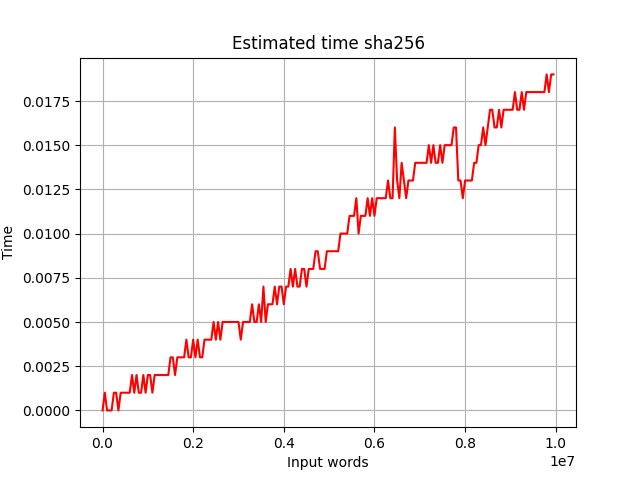
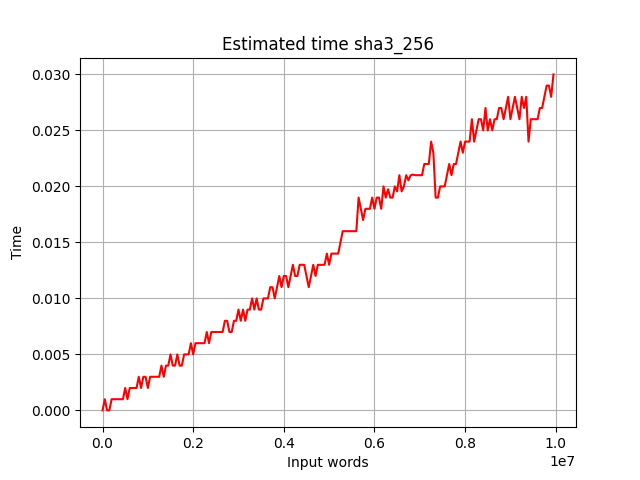
 

Рис.11 – график времени SHA256 Рис.12 – график времени SHA3\_256

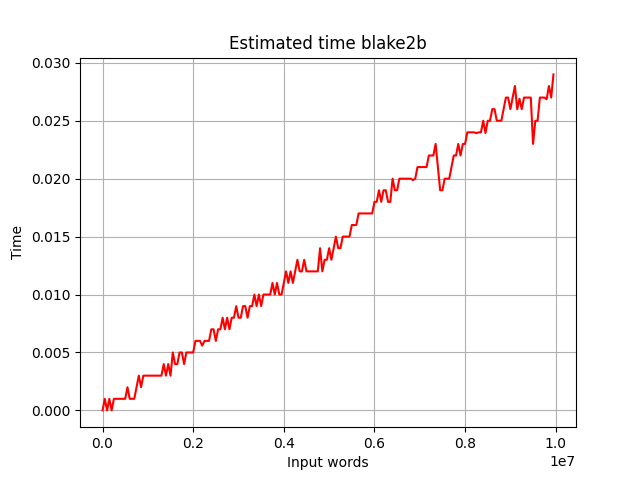
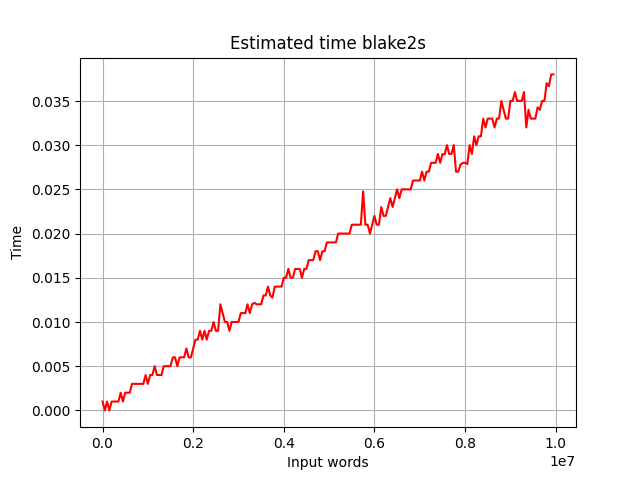
 

Рис.13 – график времени Blake2b Рис.14 – график времени Blake2s

Исходя из этих графиков, можно оценить время работы хеш-функции от длины входной строки.

# Глава 6. Код программы.

from hashlib import md5, sha1, sha256, sha3\_256, blake2b, blake2s

import time

import sys

import string

import matplotlib.pyplot as plt

def progress\_bar(iteration, total, length=40):

# Вычисляем процент завершения

percent = (iteration / total)

# Вычисляем количество символов для заполнения прогресс-бара

filled\_length = int(length \* percent)

# Создаем строку прогресс-бара

bar = '█' \* filled\_length + '-' \* (length - filled\_length)

# Выводим прогресс-бар

sys.stdout.write(f'\r|{bar}| {percent:.2%} Complete')

sys.stdout.flush()

def brute\_force(hash, total=10e7):

start = time.time()

progress\_bar(1, total)

for i in range(total):

if i %20000 == 0:

progress\_bar(i, total)

if hash == md5(str(i).encode('utf-8')).hexdigest():

print("Input password:", end = "")

print(i)

break

end = time.time()

print("Time for bruteforce" + str(end - start))

def dict\_scan(hash, file\_path):

start = time.time()

ex = dict()

k = 0

file = open(file\_path, 'r', encoding='utf-8').readlines()

total = len(file)

progress\_bar(1, total)

for i in file:

ex[md5(i[:-1].encode('utf-8')).hexdigest()] = i

k+=1

if k %20000 == 0:

progress\_bar(k, total)

mid = time.time()

try:

print('\nThe password was:'+ ex[hash])

except:

print(f"\npassword not found in dict\n Hash: {hash}")

end = time.time()

print("Time making dict:" + str(mid - start))

print("Time scan password:" + str(end - mid))

def reduce(hash\_value, length=6):

"""Функция редукции, которая преобразует хеш в строку фиксированной длины."""

# Преобразуем хеш в число

num = int(hash\_value, 16)

# Доступные символы для генерации паролей

characters = string.digits + string.ascii\_letters

# Генерация строки фиксированной длины

reduced = ''

while len(reduced) < length:

# Получаем индекс символа из числа

index = num % len(characters)

reduced += characters[index]

num //= len(characters)

return reduced

def line\_attempt(inp, attempt):

for i in range(attempt):

hash = md5(inp.encode('utf-8')).hexdigest()

inp = reduce(hash)

return inp

def rainbow\_scan(hash, file\_path, attempt=2):

start = time.time()

ex = dict()

k = 0

file = open(file\_path, 'r', encoding='utf-8').readlines()

total = len(file)

progress\_bar(1, total)

for i in file:

ex[md5(line\_attempt(i[:-2], attempt).encode('utf-8')).hexdigest()] = i[:-2]

k+=1

if k %20000 == 0:

progress\_bar(k, total)

mid = time.time()

for i in range(10):

if hash in ex.keys():

inp = line\_attempt(ex[hash], attempt-i)

print('\nThe Password was: ' + inp)

end = time.time()

print("Time making rainbow:" + str(mid - start))

print("Time scan password:" + str(end - mid))

return 1

else:

hash = md5(reduce(hash).encode('utf-8')).hexdigest()

print('\nThe Password not found')

end = time.time()

print("Time making rainbow:" + str(mid - start))

print("Time scan password:" + str(end - mid))

def test\_hash\_time(file\_path, hash):

file = open(file\_path, 'r', encoding='utf-8').readlines()

total = len(file)

s = ""

k = 0

res = 200

times = []

s = ' '.join(list(file))

progress\_bar(0, res)

k = 0

for i in range(total//res, total+1, total//res):

st = time.time()

hash(s[0:i].encode('utf-8'))

end = time.time()

times.append(end-st)

progress\_bar(k, res)

k+=1

progress\_bar(res, res)

fig = plt.figure()

plt.plot([i\*total//res for i in range(len(times))], times, 'r')

a = hash('111'.encode('utf-8'))

#plt.scatter([i\*total//res\*3 for i in range(len(times[::3]))], times[::3], marker='\*')

plt.title(f"Estimated time {a.name}")

plt.xlabel('Input words')

plt.ylabel('Time')

plt.grid()

plt.show()

def main():

print("Input password: ", end = "")

password = input().encode('utf-8')

hash = str(md5(password).hexdigest())

#brute\_force(hash, 10000000)

#dict\_scan(hash,'C:\\Users\\user\\Desktop\\1234\\ignis-10M.txt')

#rainbow\_scan(hash, 'C:\\Users\\user\\Desktop\\1234\\ignis-10M.txt')

test\_hash\_time('C:\\Users\\user\\Desktop\\1234\\ignis-10M.txt', md5)

test\_hash\_time('C:\\Users\\user\\Desktop\\1234\\ignis-10M.txt', sha1)

test\_hash\_time('C:\\Users\\user\\Desktop\\1234\\ignis-10M.txt', sha256)

test\_hash\_time('C:\\Users\\user\\Desktop\\1234\\ignis-10M.txt', sha3\_256)

test\_hash\_time('C:\\Users\\user\\Desktop\\1234\\ignis-10M.txt', blake2b)

test\_hash\_time('C:\\Users\\user\\Desktop\\1234\\ignis-10M.txt', blake2s)

main()

# Глава 7. Заключение.

В данной курсовой работе были рассмотрены алгоритмы хеш-функций. Для проверки их криптостойкости были реализованы три метода атак на пароли, представленных в виде хеша. Были показаны время работ разных хеш-функций в зависимости от длины входной строки.

Анализируя проделанной выше работы, можем сделать выводы по улучшению защиты паролей для пользователей: увеличение длинны пароля, избегание существующих слов и распространенных комбинаций, использование спецсимволов и символов в разных регистрах. Также есть способы для увеличения криптостойкости путем добавления «Соли» (добавление уникальности пароля – логин)

Программа загружена на githu: …

# Глава 8. Список используемых источников

# <https://natmchugh.blogspot.com/2015/02/create-your-own-md5-collisions.html>

# <https://www.weakpass.com/wordlists>