**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

**“Большие данные и распределенная цифровая платформа”**

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Алгоритмы и структуры данных»**

**на тему «Исследование хеш-функций с различными вводными условиями»**

**Вариант – 9**

**Студент гр. 23Б15-пу**

**Сурин И.С.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2024 г.**

Оглавление

[Цель работы 3](#_Toc180757211)

[Теоретическая часть 3](#_Toc180757212)

[Хеш-функция 3](#_Toc180757213)

[Техника взлома Brute Force 4](#_Toc180757214)

[Соль 4](#_Toc180757215)

[Описание схемы пошагового выполнения алгоритма 5](#_Toc180757216)

[Описание задачи 6](#_Toc180757217)

[Основные шаги программы 6](#_Toc180757218)

[Описание программы 7](#_Toc180757219)

[Рекомендации для пользователя 7](#_Toc180757220)

[Рекомендации для программиста 8](#_Toc180757221)

[Исходный код программы 8](#_Toc180757222)

[Контрольный пример 8](#_Toc180757223)

[Анализ 11](#_Toc180757224)

[Вывод 11](#_Toc180757225)

[Источники 12](#_Toc180757226)

# **Цель работы**

Цель работы заключается в расшифровке набора данных, которые были зашифрованы с использованием хеш-функции с добавлением соли (модификатора входа). Помимо этого, важно провести анализ процесса расшифровки и изучить, как изменяются результаты в зависимости от различных условий, таких как использование различных хеш-функций SHA-1 и SHA-256, наличие или отсутствие соли, а также маски для перебора.

# **Теоретическая часть**

## **Хеш-функция**

Хеш-функция — это алгоритм, который преобразует входные данные произвольной длины в фиксированное значение — хеш. Результатом работы хеш-функции является уникальная последовательность символов, которая служит идентификатором исходных данных. Основные характеристики хеш-функций:

1. **Односторонность**: преобразование данных в хеш легко, но восстановить исходные данные по хешу практически невозможно.
2. **Фиксированная длина**: независимо от длины исходных данных, хеш-функция всегда возвращает хеш фиксированной длины (например, 160 бит для SHA-1 или 256 бит для SHA-256).
3. **Уникальность**: малейшее изменение исходных данных приведет к значительному изменению хеша.
4. **Детерминированность**: одно и то же значение всегда приводит к одинаковому хешу.

Хеш-функции широко используются для хранения паролей, проверок целостности данных и в криптографии. Однако они уязвимы к атакам типа Brute Force.

Примеры популярных хеш-функций:

* **SHA-1** — генерирует хеш длиной 160 бит.
* **SHA-256** — улучшенная версия SHA-1 с хешем длиной 256 бит.

## **Техника взлома Brute Force**

Brute Force (перебор) — это метод атаки, при котором программа последовательно проверяет все возможные комбинации символов для восстановления исходного значения, соответствующего хешу. Этот метод не требует знаний о структуре данных, но он крайне медленный для сложных паролей или длинных данных.

Основные этапы brute force-атаки:

1. **Определение диапазона значений**: программа формирует все возможные комбинации символов, которые могут быть использованы в пароле (например, цифры, буквы и специальные символы).
2. **Генерация хешей**: для каждой возможной комбинации вычисляется хеш.
3. **Сравнение хеша**: программа сравнивает полученный хеш с целевым хешем. Если хеши совпадают, исходное значение восстановлено.

Для повышения эффективности взлома используют различные стратегии:

* **Атака по маске**: если известен формат пароля, можно ограничить диапазон поиска (например, если пароль состоит только из цифр).
* **Словарные атаки**: вместо полного перебора, проверяются значения из заранее составленного списка (словаря) вероятных паролей.

Главный недостаток техники Brute Force — её время выполнения, которое растет экспоненциально с увеличением длины пароля и числа возможных символов.

## **Соль**

Соль (salt) — это случайное значение, добавляемое к данным перед их хэшированием. Цель соли — усложнить атаки на хеши, такие как Brute Force. Без соли одинаковые значения всегда приводят к одинаковым хешам, что делает их уязвимыми для атак. Соль позволяет этого избежать, даже если несколько пользователей используют одинаковые пароли.

Принцип работы соли:

1. При хэшировании данных (например, пароля) к ним добавляется случайное значение (соль).
2. Полученные данные с солью хэшируются, и результат сохраняется.
3. При проверке пароля соль извлекается и добавляется к введенным данным перед хэшированием для сравнения с хранимым значением.

Преимущества соли:

* **Уникальные хеши для одинаковых данных**: если два пользователя используют один и тот же пароль, их хеши будут разными благодаря соли.
* **Защита от радужных таблиц**: радужные таблицы — это предвычисленные хеши для многих значений. Использование соли делает такие таблицы неэффективными, так как каждая комбинация "соль + пароль" требует отдельного расчета.

Пример использования соли: Если пароль — "password", а соль — "12345", хешируется строка "password12345", и результат хеширования сохраняется. В дальнейшем для проверки пароля вводится пароль, добавляется соль, и вычисляется новый хеш для сравнения с хранимым значением.

# **Описание схемы пошагового выполнения алгоритма**

* 1. Загрузка зашифрованного датасета
  2. Применение алгоритмов для расшифровки данных с использованием метода Brute force
  3. Тестирование программы с помощью хеш-функции MD5, вследстиви чего получаем телефонные номера с солью
  4. Сначала создается файл с набором данных (телефонных номеров), которые будут хешироваться. Эти данные хранятся в файле corrected\_numbers.csv
  5. Программа загружает данные, обрабатывает их и сохраняет только сами номера телефонов в файле corrected\_phones.txt
  6. Хеш-функции SHA-1 и SHA-256 применяются к каждому номеру телефона из файла. На выходе получаются два массива хешей, которые записываются в отдельные файлы: sha1\_hashes\_1.txt и sha256\_hashes.txt
  7. В программе используется утилита Hashcat для попытки взлома хешей с помощью метода перебора. Она выполняет перебор числовых значений длиной 9 символов, начиная с префикса "89", что характерно для телефонных номеров
  8. Взломанные значения сохраняются в файлы cracked.txt (для SHA-1) и cracked\_sha256.txt (для SHA-256)

# **Описание задачи**

Задача состоит в расшифровке датасета с номерами телефонов, с последующим анализом подобной задачи с различными входными данными:

1. Программа должна считывать входной файл (scoring\_data\_v.1.9.xlsx) с хешами и корректрными номерами телефонов и разбить его на два файла hashes.txt и phones.txt.
2. Расшифровка датасета с последующим нахождением соли.
3. Зашифровать датасет с хэш-функциями разных семейств и протестировать расшифровку с разными солями.
4. Найти минимальное количество телефонов для нахождения соли.

# **Основные шаги программы**

1. **Запуск программы**: Пользователь запускает скрипт для считывания входного файла, и загружаются два файла hashes.txt и phones.txt.
2. **Расшифровка хэшей**: С помощью утилиты hashcat расшифровать хэши и сохранить их в файл resulst.txt.
3. **Нахождение соли**: С помощью python скрипта находится соль и сохраняется в corrected\_numbers.csv без соли, а соль идет отдельным стоблцом.
4. **Тестирование задачи**: Задача тестируется с различными солями и хеш-функциями.

# **Описание программы**

Программная реализация написана на языке Python 3.12.2 с использованием утилиты hashcat и библиотеки hashlib. Задача выполняется посредством последовательного выполнения скриптов и команд hashcat.

Таблица1 Скрипты

|  |  |  |
| --- | --- | --- |
| Скрипт | Описание | Возвращает |
| salt.py | Находит соль для расшифрованного файла с помощью известных номеров телефона и записывает сырые номера телефонов в файл. | Соль |
| help.py | Вычисляет реальные номера и записывает в corrected\_numbers.csv | corrected\_numbers.csv |
| 1.py | Хеширует телефоны различными функциями | sha1\_hashes\_1.txt  sha256\_hashes.txt |
| dataset.py | Принимает входной .xlsx файл и разбивает его на два файла .txt | phones.txt, hashes.txt |

# **Рекомендации для пользователя**

1. **Использование сложных паролей и данных**: Для защиты данных (например, паролей) рекомендуется использовать сложные и длинные пароли, которые включают комбинации букв, цифр и символов. Это значительно усложняет перебор значений методом Brute force.
2. **Обновление паролей**: Регулярно обновляйте свои пароли и используйте уникальные пароли для разных сервисов. Это снизит риск взлома при утечке данных с одного ресурса.

# **Рекомендации для программиста**

1. **Использование соли при хешировании**: Соль должна быть безопасной и уникальной для каждого хешируемого значения. Это защитит от атак с использованием радужных таблиц.
2. **Держать актуальными**
3. **Безопасное хранение соли**: Убедитесь, что соль сохраняется и используется корректно. Соль можно хранить вместе с хешем, но она должна быть уникальной для каждого пользователя, чтобы атаки на один хеш не помогли взломать другие данные.
4. **Защита от атак методом Brute force**: Вводите меры защиты от попыток перебора (Brute force) паролей. Это может быть ограничение количества попыток входа, введение капчи или временная блокировка аккаунта после нескольких неудачных попыток.
5. **Регулярные аудиты безопасности**: Периодически проверяйте и обновляйте алгоритмы и параметры хеширования, чтобы они соответствовали современным стандартам безопасности.

# **Исходный код программы**

<https://github.com/Ignatio27/spbu-algorithms-and-data-structures->

# **Контрольный пример**

1. **Считывание входного файла**Для считывания входного файла запустите скрипт dataset.py.

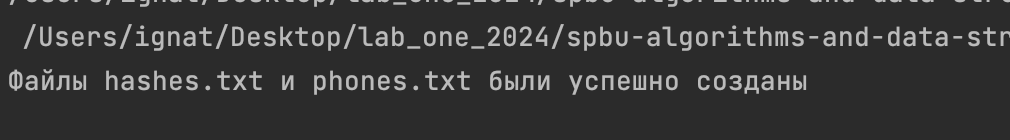
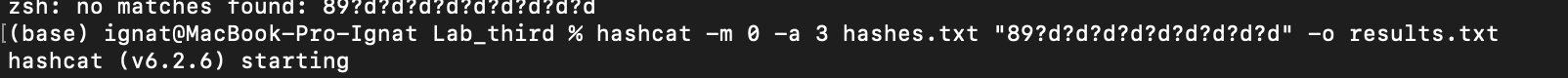
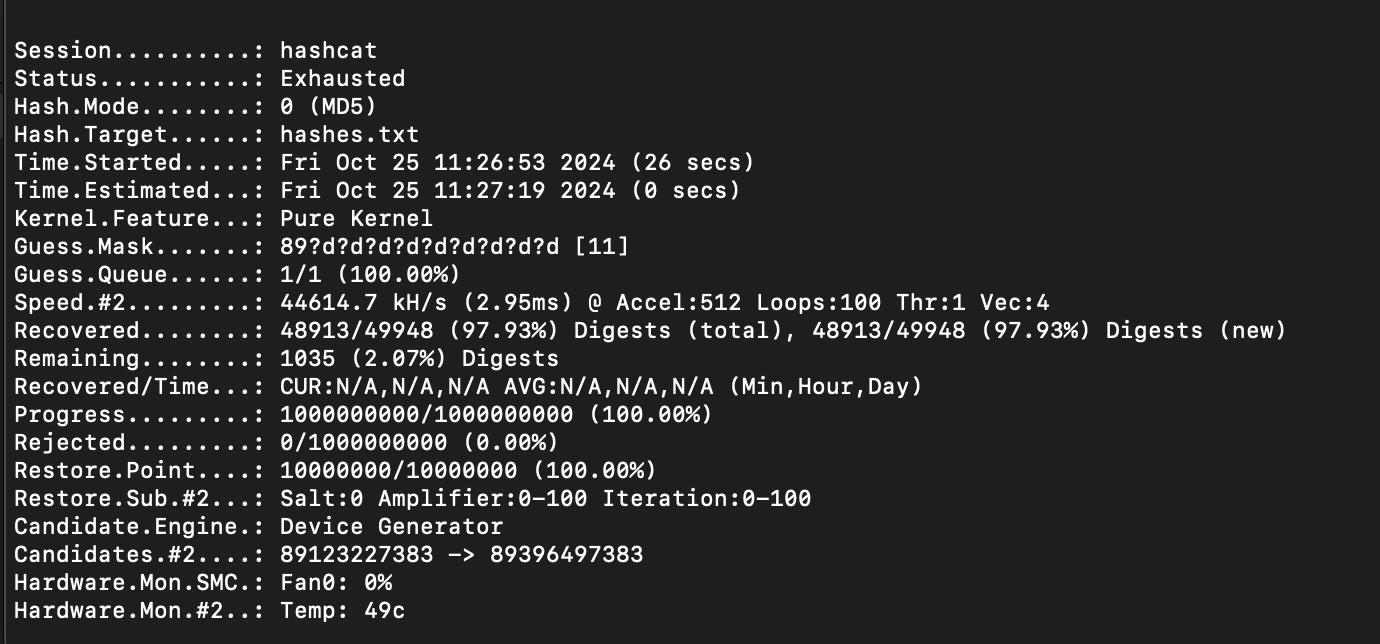


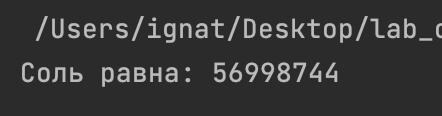
Рисунок 1. Запуск фала dataset.py

1. **Расшифровка**   
   Расшифровка хешей утилитой hashcat с помощью атаки по маске (Рис. 1 и 2).

  
Рисунок 2. Пример команды для расшифровки хэшей MD5

  
Рисунок 3. Результат расшифровки входного датасета

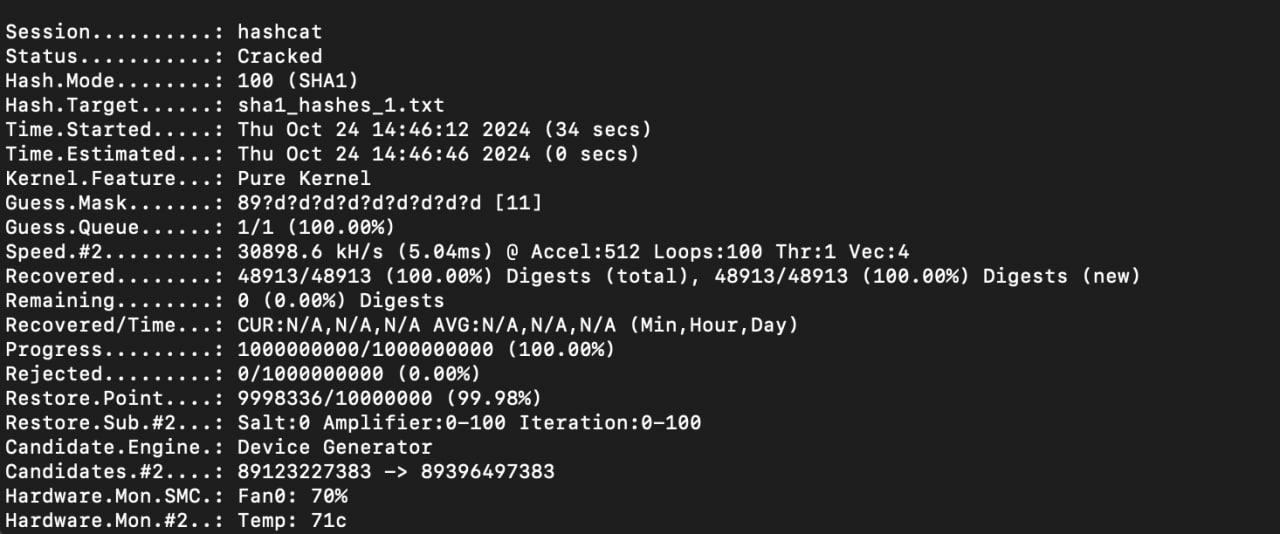
1. **Нахождение соли**Запуск скрипта salt.py, который находит соль и записывает номера телефонов без соли в отдельный файл corrected\_phones.csv.

  
Рис 4. Соль, найденная скриптом

1. **Тестирование задачи с различными солями и хеш-функциями**  
   Запуска скрипта 1.py, который хэширует исходный и сохраняет зашифрованные номера телефонов в отдельные файл. Далее с помощью утилиты hashcat происходит расшифровка хешей. (Рис. 4 – 6)



Рис 5. Вывод скрипта 1.py

  
Рис 6. Результат расшифровки датасета с хеш-функуией sha1

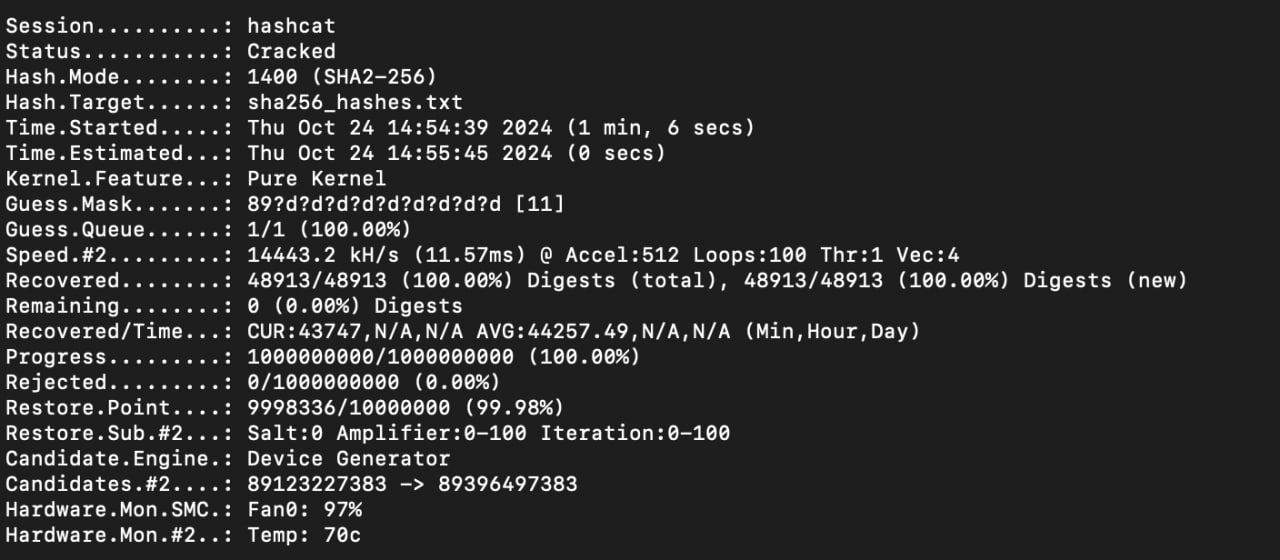
  
Рис 7. Результат расшифровки датасета с хеш-функуией sha256

Таблица 2. Анализ соли

|  |  |  |
| --- | --- | --- |
| Соль | Хэш | Время выполнения |
| 1 буква | Sha1 |  |
| 2 буквы | Sha1 |  |
| 3 буквы | Sha1 |  |
| 1 цифра | Sha1 |  |
| 2 цифры | Sha1 |  |
| 3 цифры | Sha1 |  |
| 1 буква | Sha256 |  |
| 2 буквы | Sha256 |  |
| 3 буквы | Sha256 |  |
| 1 цифра | Sha256 |  |
| 2 цифры | Sha256 |  |
| 3 цифры | Sha256 |  |

# **Анализ**

Время выполнения расшифровки хешей с помощью утилиты hashcat в основном определяется типом используемой хеш-функции. В рамках данной задачи, на скорость расшифровки не влияет длина слои, в случае сложения/вычитания соли, однако, на время выполнения влияет длина соли в случае конкатинации, также тип хеширования тоже влияет на время выполнения. Для того чтобы определить соль, необходимо минимум три телефонных номера.

# **Вывод**

В рамках данной работы был разработан алгоритм для расшифровки датасета с номерами телефонов. Реализованный алгоритм позволяет проводить анализ выполнения задачи на различных входных данных, что способствует более глубокому пониманию работы хеш-функций и метода brute-force.

# **Источники**

1. hashcat’s documentation // hashcat URL: https://hashcat.net/wiki/ (дата обращения: 24.10.2024).
2. Python hashlib documentation // hashlib URL: https://docs.python.org/3/library/hashlib.html (дата обращения: 24.10.2024).