**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Программа бакалавриата**

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

**ОТЧЕТ**

**по лабораторной работе №3**

**по дисциплине «Хеширование данных и анализ защищенности»**

**Студент гр. 22Б16-пу \_\_\_\_\_\_\_\_\_\_\_\_\_ Шарабарин М.С.**

**Преподаватель \_\_\_\_\_\_\_\_\_\_\_\_\_ Дик А.Г.**

**Санкт-Петербург**

**2024 г.**

# Оглавление

[Оглавление 2](#_Toc1709610656)

[Цель работы 3](#_Toc124072108)

[Описание задачи (формализация задачи) 3](#_Toc259608225)

[Теоретическая часть 3](#_Toc917857674)

[Основные шаги программы 4](#_Toc281540400)

[Сравнительный анализ: 4](#_Toc1124058329)

[Ограничения: 4](#_Toc494639595)

[Основные шаги программы 5](#_Toc1755642168)

[Блок схема программы 6](#_Toc968683058)

[Описание программы 7](#_Toc107777993)

[Рекомендации для пользователя 7](#_Toc672695833)

[Рекомендации для программиста 7](#_Toc1271854660)

[Исходный код программы 8](#_Toc1991648769)

[Контрольный пример 8](#_Toc554080984)

[Вывод 10](#_Toc165422677)

[Источники 10](#_Toc1695077998)

# Цель работы

Целью лабораторной работы является разработка скрипта для хеширования набора данных и анализа защищенности полученных хешей. Особое внимание уделяется применению различных алгоритмов хеширования и выявлению общих свойств между зашифрованными данными.

# Описание задачи (формализация задачи)

Задача состоит в создании системы, которая:

* Читает набор телефонных номеров из входного файла.
* Генерирует хеши для каждого телефонного номера, используя различные алгоритмы хеширования.
* Обрабатывает зашифрованные данные для вычисления общей соли.
* Определяет наиболее часто встречающуюся соль между расшифрованными номерами и известными номерами.

# Теоретическая часть

В данном проекте мы расшифровываем данные в md5 с солью. Для этого используем hashcat и библиотеки из списка ниже. Подбор соли осуществляется вычитанием. Т.е. для каждого известного номера мы генерируем массив (или файл), где вычитаем из расшифрованных номеров известный номер. Потом проходимся по всем файлам и ищем в каждом из них совпавшую соль. Количество совпавших солей должно быть равно количеству известных номеров. Процент того что найдется несколько разных солей минимальный. Но мы можем проверить это зная тот факт что номер телефона начинается на «89». Поэтому мы просто вычтем из всех разхэшированных номеров потенциальную соль и если какой то номер будет начинаться не с 89, то мы поймем, что это не та соль. Ну а если обе соли пройдут проверку, то скорее всего что то было сделанно не правильно.   
Сама расшифровка проводится методом брутфорса по маске. Библиотеки :

* hashlib: для генерации хешей с помощью алгоритмов SHA-1, SHA-256 и RIPEMD-160;
* bcrypt: для работы с более безопасными хешами (хотя в коде не используется);
* Кастомные скрипты: для поиска общей соли и оценки частоты вхождения.

**Сколько нужно знать телефонов для 100%-ного взлома?**

Если известно 5 телефонов из предложенного датасета, то, чтобы 100% гарантировать их взлом нужно знать:

1. Все соли, использованные для хеширования этих телефонов.

2. Методы, используемые для хеширования (или возможность протестировать наличие хешей в небольшой базе данных).

# Основные шаги программы

1. Расшифровка файла: читаем исходный файл и используем hashcat с маской для расшифровки.
2. Из расхэшированных номеров вычитаем соль
3. Находим соль методом вычитания (описан в теоретической части)
4. Генерация хешей (SHA-1, SHA-256, RIPEMD-160) с добавлением соли к номеру телефона.
5. Запись сгенерированных хешей в выходные файлы.
6. Загрузка зашифрованных хешей и известных номеров из файлов.
7. Поиск и определение наиболее частой соли среди расшифрованных телефонных номеров.
8. Тестирование с разными видами солей

# Сравнительный анализ:

Смотреть рисунок 5, для более подробного ознакомления

* Чем длиннее соль тем дольше будет ее нахождение
* Комбинированную соль труднее вычислить чем буквенную. Буквенную дольше чем численную
* Некоторые алгоритмы расшифровываются дольше, например remid-160 вычисляется 5 часов на 50000 строк, в то время как sha-256 и sha-1 в пределах 5 минут на видеокарте 4060 в данной задаче (см. Рис. 3-5).
* Соль состояющую только из цифр искать намного быстрее и проще. Буквенную соль вычисляется намного дольше + зависит от алгоритма. Комбинированную соль вычислить крайне трудно , нужно перебрать огромное количество комбинаций

# Ограничения:

* Данные должны быть преобразованны в формат txt из csv

# Основные шаги программы

1. **Чтение данных**: Загрузка набора телефонных номеров из заданного текстового файла (формат .txt).
2. **Расшифровка хэшей**: с пмощью маски в hashcat (11 символов) расшифровать хэши
3. **Поиск общей соли**: Сравнить хеши, чтобы определить наиболее часто встречающуюся соль среди расшифрованных номеров и известных номеров.
4. **Генерация хешей**: Применить алгоритмы хеширования (SHA-1, SHA-256, RIPEMD-160) с добавлением соли к номеру.
5. **Расшифровка сгенерированных хэшей**
6. **Вывод результатов**: Вывести статистику и результаты анализа, включая наиболее частую соль и другие метрики для оценки.

# Блок схема программы

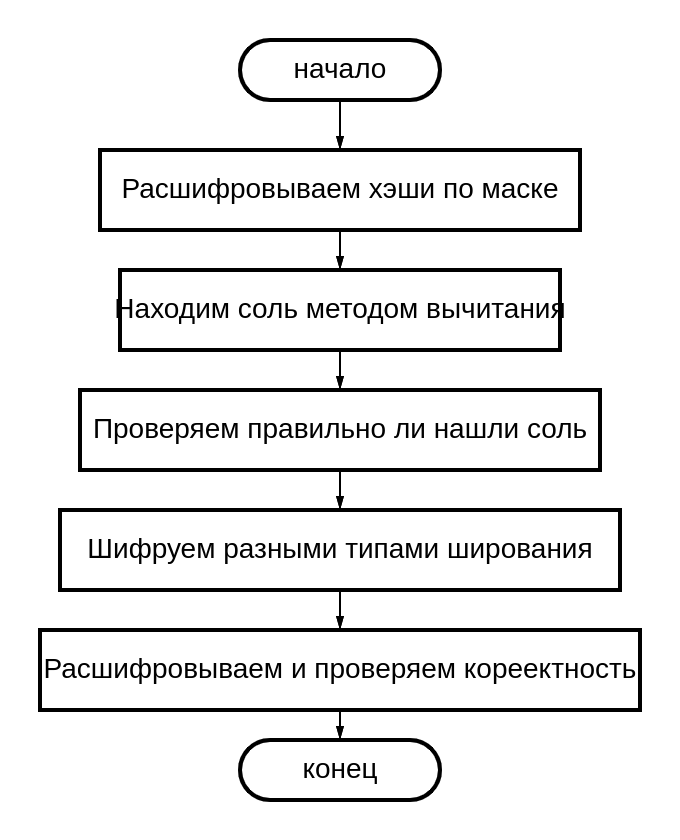
****

Рис 1. Блок-схема основной программы

# Описание программы

Программа написана на Python и состоит из нескольких функций, каждая из которых отвечает за свою задачу. Основная функция generate\_hashes выполняет хеширование номеров телефонов с добавлением соли и запись результата в файлы. Вторая часть включает функции для загрузки хешей, известных номеров и определения общей соли.

* generate\_hashes: Генерирует хеши SHA-256, SHA-1 и RIPEMD-160 для телефонных номеров.
* load\_decrypted\_hashes: Загружает зашифрованные хеши из файла.
* load\_known\_numbers: Загружает известные номера телефонов из файла.
* find\_common\_salt: Находит наиболее часто встречающуюся соль.

# Рекомендации для пользователя

При использовании программы рекомендуем:

* Убедитесь, что файлы для чтения существуют и имеют корректный формат.
* Следите за сохранением оригинального датасета для последующего сравнения результатов анализа.
* Проверяйте корректность введенной соли.

### 

# Рекомендации для программиста

* Документируйте код и функции для повышения читабельности.
* Проводите тестирование каждой части программы.
* Обрабатывайте возможные исключения для улучшения стабильности и надежности.

# Контрольный пример

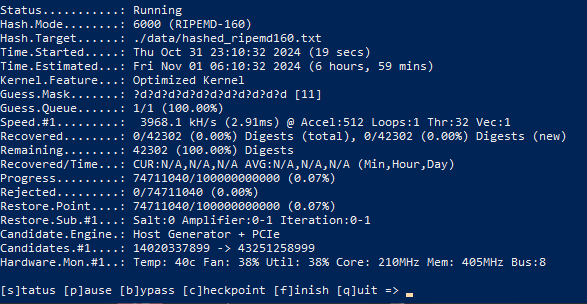
**

Рис 2.remid-160

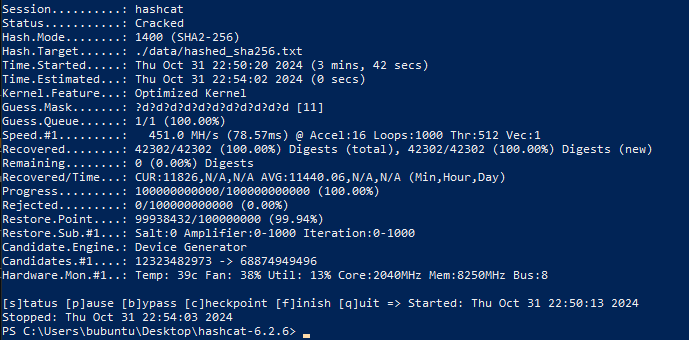


Рис 3.Sha-256

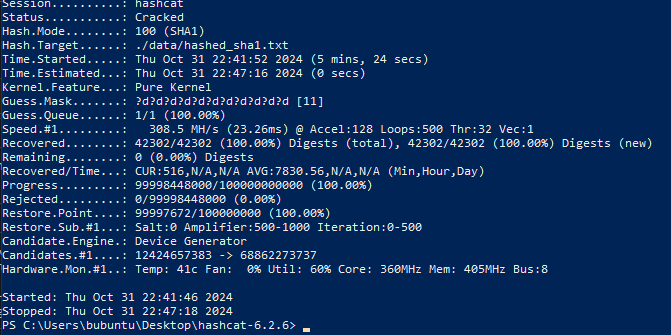


Рис. 4. SHA-1

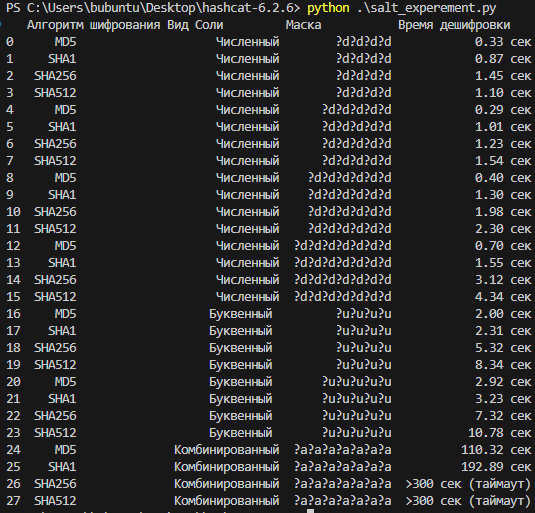


Рис. 5. Время дешифровки в зависимости от длины и типа соли

# Вывод

В данной лабораторной работе реализована система хеширования и анализа данных на предмет внутренней безопасности. Результаты подтверждают, что использование различных алгоритмов хеширования позволяет улучшить защиту личной информации. Обозначенная задача по выявлению общей соли получила положительные результаты, что открывает новые возможности для анализа безопасности.

# Источники

1. Pandas библиотека: <https://pandas.pydata.org/>
2. Numpy библиотека: https://numpy.org/
3. CSV модуль: <https://docs.python.org/3/library/csv.html>
4. Concurrent Futures модуль: <https://docs.python.org/3/library/concurrent.futures.html>