

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 – 7

**Студент гр. 23Б15-пу
Сериков К.Г.**

**Преподаватель
Дик А.Г.**

Санкт-Петербург

2024 г.

Оглавление

1. Цель работы.....	3
2. Описание задачи (формализация задачи).....	3
3. Теоретическая часть.....	4
4. Блок схема программы.....	6
5. Описание программы.....	7
6. Рекомендации пользователя.....	8
7. Рекомендации программиста.....	8
8. Исходный код программы.....	8
9. Контрольный пример.....	9
10. Анализ.....	11
11. Вывод.....	12
12. Источники.....	12

Цель работы

Целью лабораторной работы является расшифровка набора данных, зашифрованного с использованием хеш-функции и модификатора входа — соли. В ходе работы будет проведен анализ влияния различных факторов, таких как длина соли, а также выбор хеш-функции, на процесс расшифровки данных. В рамках исследования также будет оценена зависимость скорости расшифровки от этих параметров и определено, какое количество известных телефонных номеров необходимо для успешного взлома всего набора данных.

Описание задачи (формализация задачи)

Задача заключается в деобезличивании датасета, содержащего хешированные телефонные номера, с использованием соли. Основная цель — восстановить оригинальные телефонные номера путём расшифровки с учётом различных хеш-функций и параметров соли. Программа должна выполнить следующие шаги:

1. **Обработка входных данных:** Загрузить хешированные телефонные номера и соответствующие хеш-функции.
2. **Определение соли:** Используя метод полного перебора (Brute Force), выявить соль, использованную при хешировании.
3. **Расшифровка данных:** Восстановить исходные телефонные номера с учётом найденной соли.
4. **Хеширование с различными хеш-функциями:** Реализовать хеширование телефонных номеров с использованием различных алгоритмов (MD5, SHA1, BLAKE2B) и разных видов соли.
5. **Тестирование и анализ:** Провести тестирование программы на различных хеш-функциях и солях, исследуя их влияние на скорость расшифровки.
6. **Определение минимального количества известных номеров:** Оценить, сколько телефонных номеров необходимо знать для успешного взлома всего датасета.

Программа должна быть протестирована на нескольких хеш-функциях, чтобы оценить влияние этого фактора на эффективность и скорость расшифровки.

Теоретическая часть

Хеш-функция — это алгоритм, который преобразует данные произвольной длины в строку фиксированной длины, называемую хешем. Хеш-функции применяются в различных областях, таких как криптография, обеспечение целостности данных, аутентификация и хранение паролей. Главная цель хеширования — создание уникального идентификатора для данных, который невозможно восстановить в исходный вид.

Основные характеристики хеш-функций:

1. **Односторонность:** Процесс преобразования данных в хеш легко выполняется, но восстановить исходные данные по хешу крайне сложно.
2. **Фиксированная длина:** Независимо от размера входных данных, хеш всегда будет иметь одну и ту же длину (например, 160 бит для SHA-1 или 256 бит для SHA-256).
3. **Детерминированность:** Для одинаковых входных данных хеш-функция всегда будет генерировать одинаковый хеш.

Примером популярных хеш-функций являются MD5, SHA-1 и SHA-256. Однако некоторые из этих алгоритмов уже устарели и больше не рекомендуются для использования из-за уязвимостей, таких как возможность коллизий, при которых разные входные данные дают одинаковые хеши. Например, MD5 и SHA-1 считаются небезопасными, в то время как SHA-256 остаётся надежным и широко используемым.

Атака методом Brute Force

Атака методом Brute Force (перебора) — это способ поиска правильного пароля или ключа путём проверки всех возможных комбинаций. Этот метод гарантирует нахождение правильного значения, но является крайне ресурсоёмким и медленным, особенно при увеличении длины пароля. В контексте хеширования это может означать проверку всех возможных исходных данных до тех пор, пока не будет найдено совпадение с хешом. Для ускорения процесса применяются методы атаки по маске (ограничение диапазона) и словарные атаки (проверка списка вероятных значений).

Соль в хешировании

Соль — это случайная строка данных, добавляемая к исходному значению перед его хешированием. Использование соли помогает предотвратить атаки, направленные на нахождение одинаковых хешей для одинаковых входных

данных. Даже если два пользователя имеют одинаковые пароли, их хеши будут различаться, если используется уникальная соль для каждого пользователя. Это делает атаки на хешированные данные, такие как перебор или использование радужных таблиц, более сложными и ресурсоёмкими.

Использование соли значительно увеличивает безопасность хеширования, так как даже при наличии одинаковых исходных данных (например, одинаковых паролей) хеши будут уникальными. Это затрудняет работу с предварительно вычисленными таблицами и радужными таблицами, которые предназначены для ускорения взлома.

Хеширование телефонных номеров

При шифровании телефонных номеров с использованием хеш-функций необходимо учитывать особенности обработки и хранения данных. В данной работе телефоны подвергаются хешированию с использованием соли, что делает их более защищёнными от атак. Соль усложняет анализ хешированных значений, повышая безопасность данных.

Программа будет тестировать несколько хеш-функций и типов соли, чтобы оценить, как различные условия влияют на успешность и скорость атаки методом Brute Force.

Основные шаги программы

1. **Подготовка данных:** Сохранение исходных данных в два отдельных файла — `initial_hashes.txt` и `phones.txt` — для удобства работы с хешированными и оригинальными номерами.
2. **Расшифровка данных:** Использование утилиты `hashcat` для расшифровки хешированных данных и записи расшифрованных телефонных номеров в файл `cracked_hashes.txt`.
3. **Поиск соли:** Осуществление поиска соли с использованием уже известных телефонных номеров. Полученные результаты сохраняются в файл `deanonymized_phones.txt`, содержащий полностью расшифрованные данные.
4. **Хеширование данных:** Применение трёх различных хеш-функций — MD5, SHA-1 и BLAKE2b — для хеширования телефонных номеров и записи полученных хешей в файлы для дальнейшего анализа.
5. **Тестирование хешей:** Проведение атак на новые хеши для тестирования скорости и эффективности методов расшифровки, с использованием различных техник взлома.
6. **Сравнение результатов:** Сравнение и анализ полученных результатов с целью оценки эффективности различных хеш-функций и солей, а также оптимизации методов для ускоренной расшифровки.

Блок схема программы

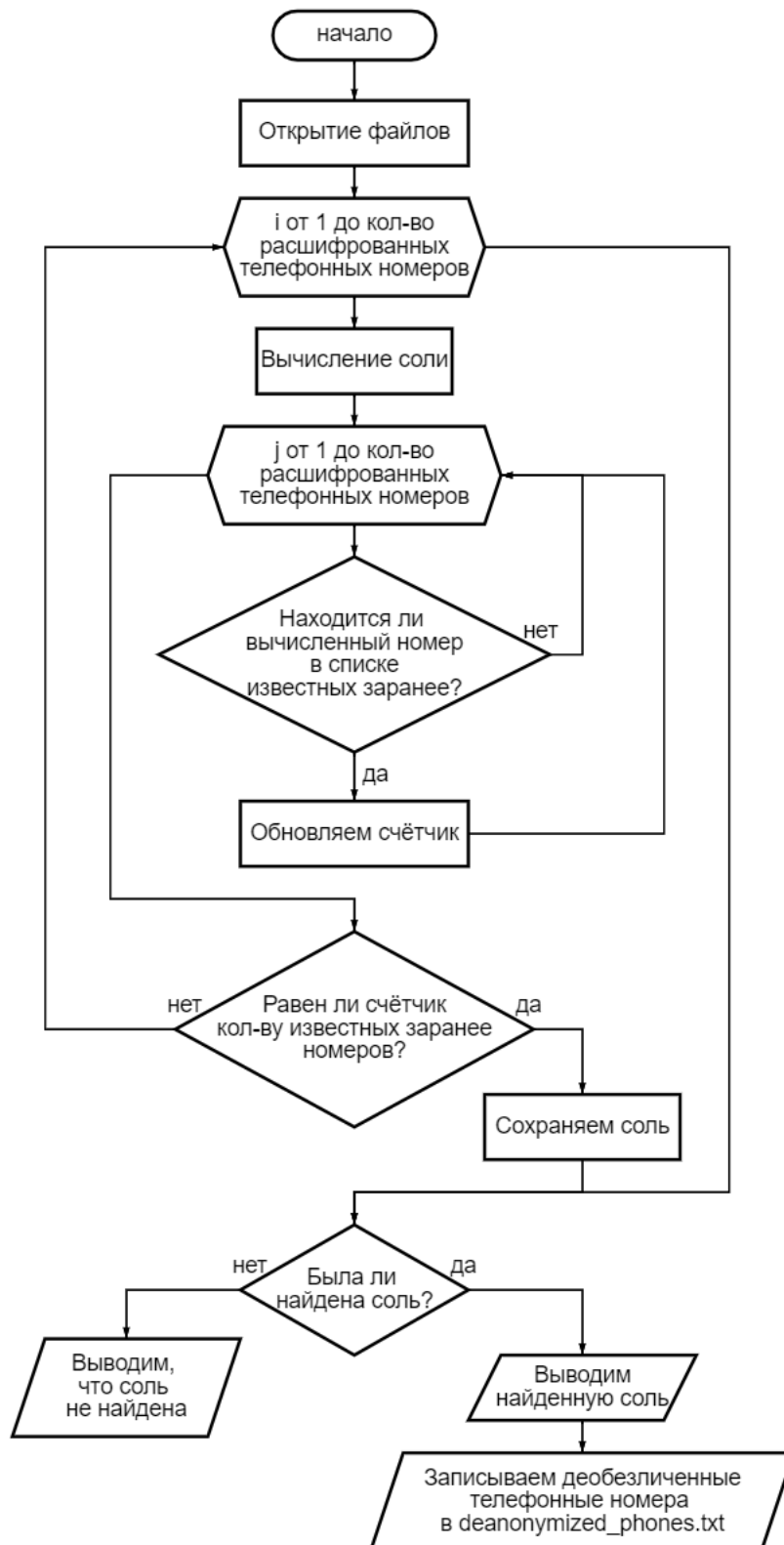


Рис 1. Блок-схема основной программы

Описание программы

Программа разработана на языке Python версии 3.13.0 и использует библиотеку `hashlib` для работы с хеш-функциями и утилиту `hashcat` для выполнения операций по взлому хешей. Основная цель программы — исследовать влияние различных условий (например, типов и длин соли, хеш-функций) на устойчивость данных к расшифровке и продемонстрировать процесс деобезличивания набора данных, зашифрованного с использованием соли.

Таблица 1. Модули

Файл	Описание
<code>phones_deanonymization</code>	Модуль для поиска соли и восстановления изначальных данных.
<code>hashing</code>	Модуль для хеширования данных с различными хеш-функциями и вариантами соли.

Основные этапы работы программы:

1. Деобезличивание — поиск соли, использованной при шифровании, и восстановление оригинальных данных. Этот процесс реализован в `phones_deanonymization.py` и включает проверку различных значений соли для поиска соответствий с данными.
2. Хеширование — создание хешей для оригинальных данных с использованием различных хеш-функций (MD5, SHA-1, BLAKE2b) и вариантов соли (малая и большая длина). Этот этап выполняется модулем `hashing.py`, который сохраняет результаты хеширования в соответствующие файлы.

Рекомендации пользователя

1. Деобезличивание:

Запустите скрипт `phones_deanonimization.py`. Выберите файл с хешами и укажите место для сохранения результата. Нажмите "Запустить Hashcat", чтобы расшифровать данные и сохранить результат.

2. Поиск соли:

После расшифровки выберите файл с результатами и нажмите "Найти и применить соль", чтобы восстановить исходные номера.

3. Хеширование:

Запустите скрипт `hashing.py`. Выберите файл с номерами телефонов и укажите место для сохранения хешей. Выберите метод хеширования (MD5, SHA-1, SHA-256) для генерации хешей.

Рекомендации программиста

Убедитесь, что установлена версия Python не ниже 3.12.0 и необходимые библиотеки: `pandas`, `tkinter`, `hashlib`.

- Проверьте, что все файлы программы находятся в одной директории.
- Скачайте и распакуйте Hashcat в директорию проекта, назвав папку "hashcat".
- Протестируйте модули с различными файлами и форматами данных для проверки корректности всех функций: расшифровка, хеширование с разными алгоритмами и применением соли.

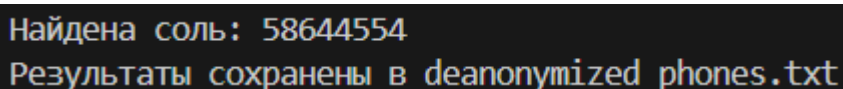
Исходный код программы

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

Контрольный пример

1. Запуск программы

Для начала работы введите в терминал команду `hashcat «hashcat -m 0 -a 3 -O --opencl-device-types 2 -w 3 initial_hashes.txt 89?d?d?d?d?d?d?d?d -o cracked_hashes.txt»`. Эта команда пытается подобрать комбинации чисел, соответствующие хешам в файле `initial_hashes.txt`, и записывает результат в `cracked_hashes.txt`. Затем запустите `«phones_deanonimization.py»`. Это программа выведет найденную соль и запишет расшифрованные данные в `«deanonimizes_phones.txt»` (Рис. 2).

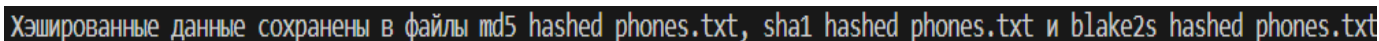


```
Найдена соль: 58644554
Результаты сохранены в deanonymized phones.txt
```

Рис 2. вывод «phones_deanonimization.py»

2. Хеширование

Для хеширования расшифрованных данных надо запустить `«hashing.py»`. Этот скрипт выполнит хеширование с выбранными алгоритмами и добавлением соли (при необходимости), сохраняя результат в указанном файле (Рис. 3).



```
Хэшированные данные сохранены в файлы md5_hashed_phones.txt, sha1_hashed_phones.txt и blake2s_hashed_phones.txt
```

Рис 3. вывод «hashing.py»

3. Анализ

Для анализа времени расшифровки получившегося файла надо открыть терминал и ввести команды: `«hashcat -m 0 -a 3 -O --opencl-device-types 2 -w 3 x_hashes.txt 89?d?d?d?d?d?d?d?d -o x_cracked_hashes.txt»`, в случае если соль — это одна цифра, и `«hashcat -m 0 -a 3 -O --opencl-device-types 2 -w 3 x_hashes.txt 89?d?d?d?d?d?d?d?d -o x_cracked_hashes.txt»`, в случае если соль — это три цифры.

В ходе выполнения этих команд `hashcat` выведет статус, где можно увидеть предполагаемое время, требуемое для расшифровки, что позволит оценить сложность атаки в зависимости от длины соли (Рис. 4 и Рис. 5).

```

Session.....: hashcat
Status.....: Running
Hash.Mode.....: 0 (MD5)
Hash.Target.....: md5_large_salt_hashes.txt
Time.Started.....: Fri Nov 08 18:34:26 2024 (22 secs)
Time.Estimated...: Fri Nov 08 21:44:37 2024 (3 hours, 9 mins)
Kernel.Feature...: Optimized Kernel
Guess.Mask.....: 89?d?d?d?d?d?d?d?d?d?d [14]
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 87696.9 kH/s (70.05ms) @ Accel:64 Loops:100 Thr:64 Vec:4
Recovered.....: 1755/48905 (3.59%) Digests (total), 1755/48905 (3.59%) Digests (new)
Remaining.....: 47150 (96.41%) Digests
Recovered/Time...: CUR:N/A,N/A,N/A AVG:N/A,N/A,N/A (Min,Hour,Day)
Progress.....: 1140326400/1000000000000 (0.11%)
Rejected.....: 0/1140326400 (0.00%)
Restore.Point....: 11403264/100000000000 (0.11%)
Restore.Sub.#1...: Salt:0 Amplifier:0-100 Iteration:0-100
Candidate.Engine.: Device Generator
Candidates.#1....: 89129422330123 -> 89396726466999

```

Рис 4. Пример вывода для команды «hashcat -m 100 -a 3 -O --opencl-device-types
2 -w 3 md5_large_salt_hashes.txt 89?d?d?d?d?d?d?d?d?d?d -o
md5_cracked_large_salt_hashes.txt»

```

Session.....: hashcat
Status.....: Quit
Hash.Mode.....: 100 (SHA1)
Hash.Target.....: sha1_large_salt_hashes.txt
Time.Started.....: Fri Nov 08 18:52:58 2024 (3 mins, 39 secs)
Time.Estimated...: Sat Nov 09 14:10:20 2024 (19 hours, 13 mins)
Kernel.Feature...: Optimized Kernel
Guess.Mask.....: 89?d?d?d?d?d?d?d?d?d?d [14]
Guess.Queue.....: 1/1 (100.00%)
Speed.#1.....: 14401.4 kH/s (62.40ms) @ Accel:32 Loops:100 Thr:64 Vec:4
Recovered.....: 20930/48905 (42.80%) Digests (total), 20930/48905 (42.80%) Digests (new)
Remaining.....: 27975 (57.20%) Digests
Recovered/Time...: CUR:6769,N/A,N/A AVG:5743.16,N/A,N/A (Min,Hour,Day)
Progress.....: 3086745600/1000000000000 (0.31%)
Rejected.....: 0/3086745600 (0.00%)
Restore.Point....: 30670848/100000000000 (0.31%)
Restore.Sub.#1...: Salt:0 Amplifier:0-100 Iteration:0-100
Candidate.Engine.: Device Generator
Candidates.#1....: 89127099534345 -> 89398403634345

```

Рис 5. Пример вывода для команды «hashcat -m 100 -a 3 -O --opencl-device-types
2 -w 3 sha1_large_salt_hashes.txt 89?d?d?d?d?d?d?d?d?d?d -o
sha1_cracked_large_salt_hashes.txt»

Анализ

Скорость расшифровки данных, например, при атаке на хеши, зависит от трёх основных факторов: вида соли, длины соли, и используемой хеш-функции.

Вид соли

Соль — это случайная строка, добавляемая к данным перед их хешированием, чтобы усложнить процесс взлома. Различные виды соли создают разную степень сложности для атаки:

- Цифровая соль (только цифры): имеет всего 10 вариантов для каждой позиции, что делает перебор комбинаций относительно быстрым.
- Буквенная соль (строчные и прописные буквы): увеличивает сложность до 52 вариантов на позицию.

Длина соли

Чем больше длина соли, тем больше возможных уникальных комбинаций. Увеличение длины соли экспоненциально повышает количество комбинаций, что напрямую влияет на время, необходимое для подбора хеша. Например:

- Для соли длиной 4 символа (только цифры) потребуется проверить $10^4 = 10,000$ комбинаций.
- Для соли длиной 4 символа (буквы, цифры и спецсимволы) потребуется проверить $94^4 = 78,074,896$ комбинаций.

Хеш-функция

Тип хеш-функции играет ключевую роль в определении времени, необходимого на каждый цикл перебора.

- Быстрые хеш-функции (например, MD5 или SHA-1): генерируют хеши за короткий промежуток времени, что упрощает атаки брутфорс. Они менее устойчивы к современным угрозам, особенно в сочетании с простой солью.
- Медленные хеш-функции (например, bcrypt, scrypt): специально разработаны для замедления процесса хеширования. Они требуют больше вычислительных ресурсов (времени и памяти), что значительно увеличивает время атаки, даже если соль проста.

Анализ на практике

Проведем анализ зависимости времени расшифровки хэшей от типа алгоритма, типа соли и длины соли на практике.

Таблица 2. Анализ расшифровки

Алгоритм	Тип соли	Длина соли	Время
MD5	Цифры	1	8 мин.
MD5	Цифры	3	3 ч
MD5	Буквенная	1	10 мин.
SHA1	Цифры	1	10 мин.
SHA1	Цифры	3	19 ч.
SHA1	Буквенная	1	15 мин.
BLAKE2B	Цифры	1	1 ч.
BLAKE2B	Цифры	3	26 д.
BLAKE2B	Буквенная	1	1.5 ч.

На основании таблицы можно сделать несколько выводов.

1. **Увеличение длины соли** заметно повышает время, необходимое для расшифровки. Это происходит из-за возрастания количества возможных комбинаций, которые требуется протестировать. Например, увеличение длины соли с 1 до 3 цифр увеличивает время расшифровки для MD5 с минут до нескольких часов, а для более сложных алгоритмов, таких как BLAKE2B, до нескольких дней.
2. **Тип соли также влияет на скорость атаки.** Цифровая соль быстрее для перебора, поскольку в ней всего 10 символов на позицию, тогда как буквенная соль добавляет больше вариантов (52 символа). В случае MD5 и SHA1, буквенная соль даже при длине в 1 символ увеличивает время расшифровки на несколько минут по сравнению с цифровой солью, а для BLAKE2B — на полчаса. Это показывает, что соль с более широким набором символов добавляет устойчивость к атаке, особенно в сочетании с более сложными алгоритмами.
3. **Сложность алгоритма хеширования также имеет значение.** Например, BLAKE2B требует больше времени для подбора хеша по сравнению с MD5 и SHA1. В нашем анализе BLAKE2B с 3-значной цифровой солью занимает до нескольких дней, что делает его значительно более устойчивым к атакам, чем MD5 и SHA1.

Вывод

В ходе лабораторной работы было проведено исследование хеш-функций с использованием соли для защиты и последующего восстановления телефонных номеров. На первом этапе изучили особенности хеширования и разработали программу для деанонимизации данных. Программа была протестирована на исходном датасете и на нескольких других алгоритмах хеширования, включая MD5, SHA-1 и BLAKE2B, что позволило провести сравнительный анализ их эффективности.

Анализ показал, что время, необходимое для восстановления данных, зависит от типа и длины соли, а также от выбранного алгоритма хеширования. Например, применение более длинной соли значительно увеличивает трудоемкость взлома, повышая стойкость хеша. Тестирование продемонстрировало, что использование хеширования с солью эффективно защищает личные данные. Однако выявились и уязвимости: знание нескольких номеров из датасета может облегчить восстановление исходных данных, особенно при слабом алгоритме или короткой соли.

Источники

1. Python hashlib documentation // hashlib URL: <https://docs.python.org/3/library/hashlib.html> (дата обращения: 25.10.2024).
2. hashcat's documentation // hashcat URL: <https://hashcat.net/wiki/> (дата обращения: 26.10.2024).
3. Хеш-функция // wiki URL: https://ru.wikipedia.org/wiki/Хеш-функция#Криптографическая_соль (дата обращения: 26.10.2024).