Санкт-Петербургский государственный университет

Группа 23.Б16-пу

Лабораторная работа по дисциплине
Алгоритмы и структуры данных

«Исследование хеш-функций с различными вводными условиями»

Выполнил студент:
Зайнуллин Мансур Альбертович

Преподаватель:
Дик Александр Геннадьевич
ассистент кафедры компьютерного моделирования
и многопоточных систем

Санкт-Петербург
2024

Оглавление

1 Цель работы **2**

2 Основные понятия и методы шифрования данных **3**

2.1 Хеш-функции 3

2.2 Метод Brute Force 3

2.3 Соль и её роль в защите данных 4

3 Описание схемы пошагового выполнения алгоритма **5**

4 Формализация задачи, представление программы на языке Python и спецификация программы **10**

4.1 Формализация задачи 10

4.2 Основные этапы работы программы 10

4.3 Спецификация программы 11

5 Контрольный пример **14**

5.0.1 Установка и настройка окружения 14

5.0.2 Запуск программы и выполнение задач 15

5.1 Шаги работы с программой 15

5.2 Ожидаемые результаты 16

6 Анализ результатов работы алгоритма и вводных условий **17**

6.1 Влияние выбора хеш-функции 17

6.2 Влияние вида соли 17

6.3 Влияние длины соли 18

6.4 Влияние структуры и объема данных 18

6.5 Заключение по результатам 19

7 Вывод **20**

1 Цель работы

Цель работы — изучить методы расшифровки данных, зашифрованных хеш-функцией с использованием соли. Для этого требуется создать программу для деобезличивания данных, проанализировать влияние типа и длины соли, а также алгоритма хеширования на скорость расшифровки.

2 Основные понятия и методы шифрования данных

В данной лабораторной работе для защиты данных используется метод хеширования с добавлением соли. Этот раздел кратко описывает ключевые понятия, необходимые для понимания задачи: хеш-функции, метод Brute Force и роль соли в защите данных.

2.1 Хеш-функции

Хеш-функция — это алгоритм, который преобразует произвольные данные в строку фиксированной длины, называемую хешем. Основное свойство хеш-функции — необратимость: по полученному хешу невозможно восстановить исходные данные. Это делает хеширование полезным для хранения и передачи данных, которые нужно защитить от несанкционированного доступа. Среди наиболее распространенных хеш-функций, которые используются для защиты данных, — SHA-1, SHA-256 и SHA-512. Каждая из этих функций преобразует данные с определенной скоростью и уровнем защиты.

Основные свойства хеш-функций, которые делают их полезными для шифрования данных:

1. **Необратимость** — по хешу невозможно восстановить оригинальные данные.
2. **Высокая скорость вычислений** — функция должна быстро обрабатывать большие объемы данных.
3. **Устойчивость к коллизиям** — разные исходные данные не должны давать один и тот же хеш.

2.2 Метод Brute Force

Brute Force — это метод, при котором программа перебирает все возможные комбинации исходных данных, пока не найдет те, которые дают

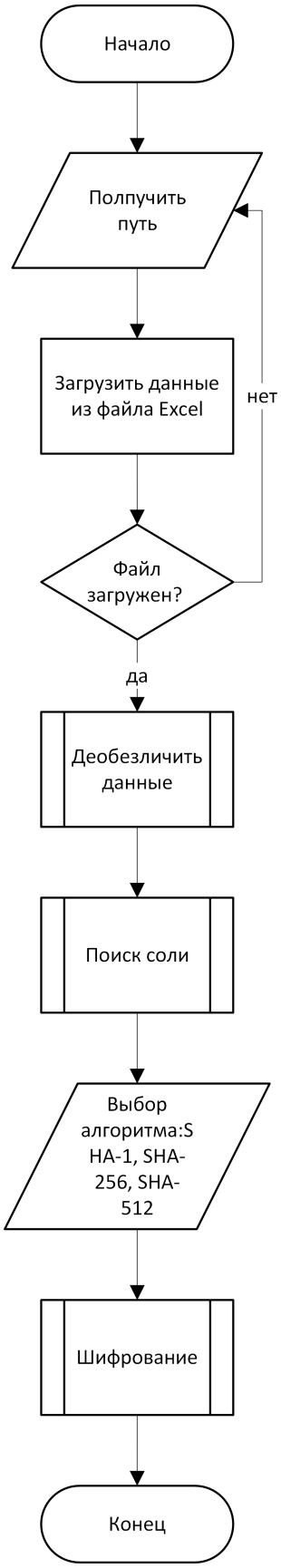
нужный хеш. Этот метод позволяет восстановить данные, но требует значительных ресурсов. Чем больше возможных комбинаций, тем больше времени и ресурсов потребуется для их перебора. Поэтому Brute Force эффективен только для относительно коротких данных. Если же данные зашифрованы с использованием соли, Brute Force становится еще более трудоемким.

2.3 Соль и её роль в защите данных

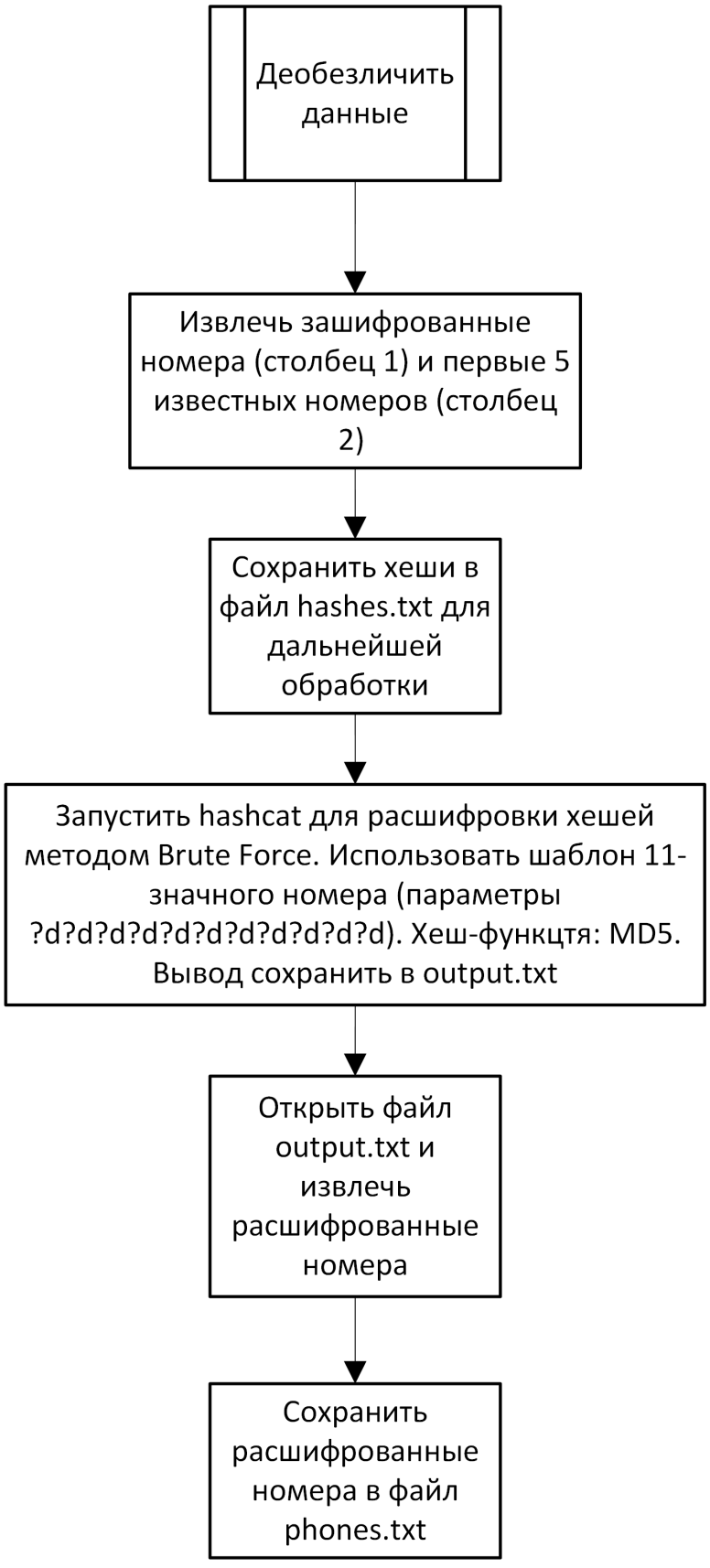
Соль — это случайное значение, которое добавляется к данным перед их хешированием. При добавлении соли даже одинаковые данные (например, одинаковые пароли или номера) при хешировании создадут разные хеши. Это затрудняет взлом, так как делает невозможным использование заранее предвычисленных таблиц (например, rainbow tables). Даже если у двух пользователей совпадают данные, соль делает каждый хеш уникальным.

Соль значительно усложняет применение Brute Force, так как для каждой комбинации исходных данных требуется учесть добавленную соль. Это увеличивает время и ресурсы, необходимые для восстановления данных, и делает атаку на зашифрованные данные более затратной.

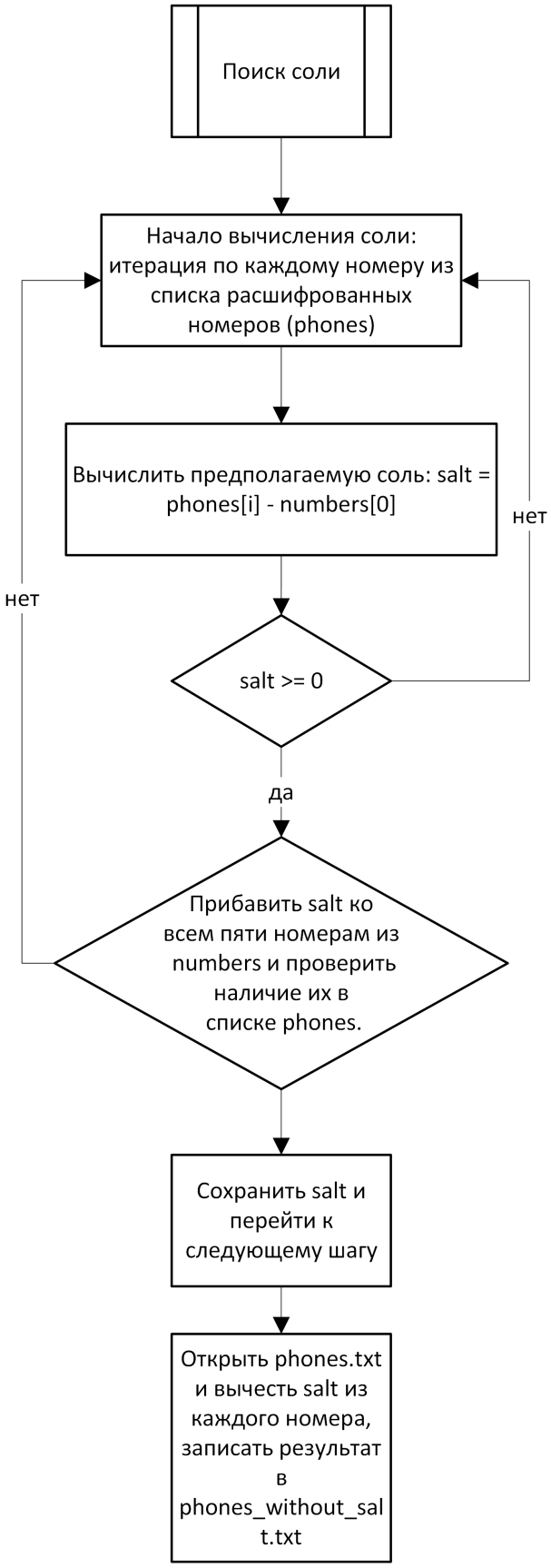
3 Описание схемы пошагового выполнения алгоритма



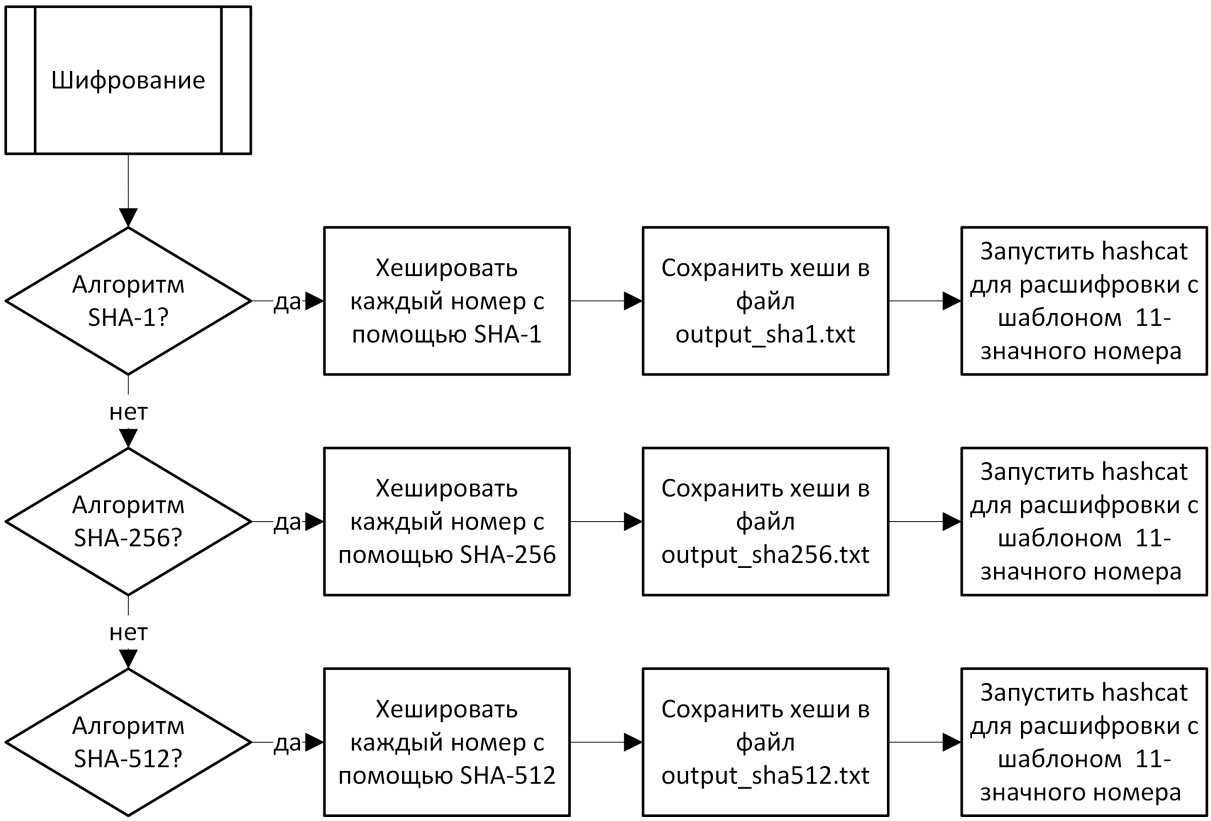
(рис. 1 Блок-схема)



(рис. 2 Деобезличить данные)



(рис. 3 Поиск соли)



(рис. 4 Шифрование)

4 Формализация задачи, представление программы на языке Python и спецификация программы

4.1 Формализация задачи

Задача программы — восстановить зашифрованные телефонные номера из набора данных, используя хеш-функции и значение соли. Программа тестирует, как разные алгоритмы хеширования и параметры соли влияют на эффективность расшифровки. Реализация задачи предполагает загрузку данных, нахождение соли, расшифровку данных и управление процессом через графический интерфейс.

4.2 Основные этапы работы программы

1. **Загрузка данных**:
    - Программа позволяет пользователю загрузить файл с данными (в формате Excel), который содержит зашифрованные телефонные номера.
    - Данные из файла сохраняются для последующего восстановления, а кнопка для деобезличивания данных становится активной.
2. **Определение соли**:
    - Программа использует функцию `compute_salt`, чтобы найти значение соли, необходимое для корректного восстановления телефонных номеров. Для этого она берет часть известных номеров, сравнивает их с зашифрованными номерами и вычисляет значение соли, которое обеспечивает совпадение.
    - После вычисления соли программа сохраняет это значение для использования на следующих этапах расшифровки данных.

3. **Деобезличивание данных (расшифровка)**:

    - Для восстановления исходных данных программа использует утилиту `hashcat`, которая выполняет Brute Force атаку для расшифровки хешей.
    - Программа формирует файл `hashes.txt`, содержащий хешированные данные, затем запускает `hashcat` для перебора 11-значных чисел, которые соответствуют формату телефонных номеров.
    - Результаты расшифровки сохраняются в файле `phones.txt`, который затем используется для последующего удаления соли и получения исходных номеров.

4. **Шифрование данных**:

    - Программа предоставляет возможность зашифровать данные с использованием одного из алгоритмов (SHA-1, SHA-256 или SHA-512). Выбранный пользователем алгоритм используется для хеширования номеров, а результаты сохраняются в текстовый файл (`sha1.txt`, `sha256.txt` или `sha512.txt`).
    - Далее `hashcat` с соответствующими параметрами может быть запущен для расшифровки данных и анализа скорости работы каждого алгоритма.

5. **Использование графического интерфейса (GUI)**:

    - Программа предоставляет пользователю интерфейс, где кнопки управляют процессами загрузки данных, вычисления соли, расшифровки и шифрования данных. GUI объединяет все этапы работы программы, облегчая взаимодействие с данными.

4.3 Спецификация программы

1. **Использование `hashcat` и параметры командной строки**:

- Программа использует `hashcat` для восстановления исходных данных путем перебора значений, соответствующих маске 11-значных телефонных номеров.
- Ключевые параметры:
  - `-a 3` (режим маски), позволяющий оптимизировать поиск, ограничив его 11-значными числами,
  - `-m` определяет используемую хеш-функцию (`-m 100` для SHA-1, `-m 1400` для SHA-256, `-m 1700` для SHA-512),
  - `-o` указывает выходной файл для сохранения расшифрованных данных.
- Маска `?d`, повторенная 11 раз, позволяет `hashcat` перебирать только числовые значения, что ускоряет процесс.

2. **Описание файлов, создаваемых и используемых программой**:
   - `hashes.txt` — хранит хеши из файла данных для их дальнейшего восстановления.
   - `output.txt` — файл для сохранения расшифрованных данных, полученных в результате работы `hashcat`.
   - `phones.txt` — временное хранилище расшифрованных номеров до удаления соли.
   - `sha1.txt`, `sha256.txt`, `sha512.txt` — файлы для хранения промежуточных результатов хеширования.
   - `phones_without_salt.txt` — конечный файл, содержащий очищенные от соли восстановленные данные.
3. **Описание ключевых функций программы**:
   - `compute_salt` — вычисляет соль, необходимую для корректного восстановления, путем сопоставления известных и зашифрованных номеров.
   - `identify` — запускает `hashcat` для расшифровки, формируя файл с промежуточными данными (`phones.txt`).

- `sha1`, `sha256`, `sha512` — функции для хеширования данных и записи их в файлы.
- `find_salt` — удаляет соль из восстановленных номеров, записывая их в файл `phones_without_salt.txt`.
- `encrypt` — позволяет выбрать алгоритм для шифрования данных и создает файл с результатом.

4. **Описание работы GUI**:
    - GUI предоставляет кнопки для управления этапами работы программы: загрузкой данных, шифрованием, вычислением соли и деобезличиванием.
    - Взаимодействие через интерфейс защищает пользователя от ошибок, так как доступ к кнопкам зависит от выполнения обязательных этапов (например, нельзя выполнить расшифровку без загрузки данных).

5 Контрольный пример

Для демонстрации работы программы приведем пошаговый контрольный пример, который показывает основные функции: загрузку данных, расшифровку, нахождение соли и обратное шифрование с использованием выбранного алгоритма. Пример также включает проверку результатов и рекомендации по завершению работы.

5.0.1 Установка и настройка окружения

1. Установите **Git** и **Python 3.9**:
   - Git: `https://git-scm.com`
   - Python 3.9: `https://www.python.org`
2. Откройте терминал и клонируйте репозиторий с помощью команды:

   ```
   git clone https://github.com/MansurYa/labs-for-algorithms-an
   ```

   Это создаст копию проекта в текущем каталоге терминала.
3. Перейдите в директорию проекта:

   ```
   cd labs-for-algorithms-and-data-structures/Lab3
   ```

4. Установите все необходимые библиотеки:

   ```
   pip install -r requirements.txt
   ```

   Если возникают ошибки при установке, обновите `pip`:

   ```
   pip install --upgrade pip
   ```

5.0.2 Запуск программы и выполнение задач

1. Запустите программу с помощью Python:

```
python3 phone_deidentifier.py
```

5.1 Шаги работы с программой

1. **Загрузка данных**:
   - В окне программы нажмите «Загрузить файл» и выберите таблицу с зашифрованными телефонными номерами:

     `labs-for-algorithms-and-data-structures/Lab3/scoring`

   - Программа подтвердит успешную загрузку данных, и кнопка «Расшифровать» станет доступной.
2. **Расшифровка данных**:
   - Нажмите «Расшифровать» для запуска процесса деобезличивания данных.
   - Дождитесь завершения — это может занять некоторое время. Программа сохранит результаты в файл `phones.txt` и отобразит уведомление об окончании.
3. **Вычисление соли**:
   - Нажмите на кнопку «Вычислить соль», чтобы программа определила значение соли, использованное при шифровании номеров.
   - Ожидайте завершения процесса и появления уведомления о сохранении расшифрованных номеров в файл `phones_without_salt.txt`.
4. **Обратное шифрование данных**:

- Выберите алгоритм для хеширования, нажав на соответствующую кнопку:
  - «Зашифровать SHA-1»,
  - «Зашифровать SHA-256» или
  - «Зашифровать SHA-512».
- Программа зашифрует номера и сохранит результат в файл:
  - `output_sha1.txt`, `output_sha256.txt` или `output_sha512.txt` — в зависимости от выбранного алгоритма.
- Этот шаг позволяет протестировать разные методы хеширования и оценить их стойкость.

5.2 Ожидаемые результаты

- **Файл `phones_without_salt.txt`** — содержит расшифрованные телефонные номера, из которых была удалена соль.
- **Файлы `output_sha1.txt`, `output_sha256.txt`, `output_sha512.txt`** — содержат обратно зашифрованные номера, использованные для анализа стойкости алгоритмов SHA-1, SHA-256, SHA-512.

6 Анализ результатов работы алгоритма и вводных условий

В данном разделе представлен анализ факторов, влияющих на скорость расшифровки данных, а также результаты работы алгоритма деобезличивания, протестированного в условиях различных хеш-функций, видов и длин соли.

6.1 Влияние выбора хеш-функции

Тип хеш-функции существенно влияет на эффективность расшифровки:

- **SHA-1**: демонстрирует наибольшую скорость подбора из-за сравнительно меньшей сложности вычислений, но имеет меньшую стойкость. Это делает SHA-1 наименее защищенной из рассмотренных функций.
- **SHA-256**: более устойчива, требуя больше вычислительных операций, что увеличивает время расшифровки. Функция снижает вероятность успешной атаки перебором и повышает общую стойкость данных.
- **SHA-512**: самый устойчивый алгоритм из исследованных. Объем вычислений для каждого хеша максимально высок, что значительно замедляет расшифровку, однако позволяет обеспечить наибольшую защищенность.

Вывод: для увеличения стойкости данных наиболее эффективна SHA-512, а для быстрого подбора хешей — SHA-1. Таким образом, для повышения защиты данных желательно выбирать более сложные хеш-функции, такие как SHA-256 или SHA-512.

6.2 Влияние вида соли

Тип используемой соли также критически влияет на возможность и время расшифровки:

- **Отсутствие соли**: позволяет использовать известные значения напрямую, ускоряя процесс. Один известный номер и его хеш дают возможность создать словарь для быстрого восстановления данных.
- **Статическая соль**: использование одного значения соли для всех данных дает возможность взлома при наличии одного известного номера, но требует подбора соли перед дальнейшим использованием для расшифровки остальных записей.
- **Динамическая соль**: при уникальной соли для каждого номера телефонные номера становятся максимально защищенными, так как для расшифровки каждой записи требуется свой процесс подбора соли, и знание одного номера не позволяет расшифровать остальные.

Вывод: отсутствие соли или статическая соль позволяет относительно быстро подобрать значения, тогда как динамическая соль повышает защиту за счет индивидуальных значений, требуя больше ресурсов и времени.

6.3 Влияние длины соли

Длина соли влияет на трудоемкость подбора значений:

- **Короткая соль**: упрощает подбор и снижает затраты на вычисления, особенно если соль статическая.
- **Длинная соль**: увеличивает общее количество возможных значений и затрудняет взлом при динамической соли, так как требует значительных ресурсов для каждого номера.

Вывод: при динамической соли увеличение длины соли повышает сложность и время подбора значений, увеличивая защиту данных. При статической соли влияние длины соли менее значимо, но тоже увеличивает общее время подбора.

6.4 Влияние структуры и объема данных

На практике структура и объем данных также играют значимую роль в эффективности расшифровки:

- **Шаблоны данных**: если структура телефонных номеров предсказуема (например, определенные префиксы), это позволяет сократить число возможных значений. Например, использование словаря для телефонных номеров, начинающихся с «89», ускоряет процесс подбора.

- **Объем данных**: при большом количестве записей процесс взлома требует значительно больше ресурсов. Наличие параллельных вычислений, таких как графические процессоры (GPU), помогает ускорить расшифровку в условиях отсутствия соли или статической соли. При динамической соли данные можно взламывать параллельно, но это увеличивает требования к оборудованию.

6.5 Заключение по результатам

Анализ показал, что скорость расшифровки данных и устойчивость к атакам зависят от нескольких ключевых факторов:

- **Тип хеш-функции** — более сложные функции, такие как SHA-256 и SHA-512, значительно увеличивают время расшифровки.

- **Вид соли** — отсутствие соли или использование статической соли позволяет быстрее взломать данные, тогда как динамическая соль обеспечивает наибольшую защиту.

- **Длина соли** — длинная соль замедляет процесс подбора, особенно в условиях динамической соли, так как увеличивает пространство поиска.

Для обеспечения высокой защиты данных целесообразно использовать хеш-функции SHA-256 или SHA-512 в сочетании с динамической длинной солью, что требует значительных ресурсов для взлома, тем самым делая расшифровку практически невозможной в реальных условиях.

7 Вывод

Работа показала, что эффективность расшифровки данных зависит от выбора хеш-функции, вида и длины соли. Чем сложнее хеш-функция и длиннее соль, тем больше вычислительных ресурсов и времени требуется для расшифровки. Статическая соль ускоряет процесс, так как позволяет применить одно значение ко всем записям. Динамическая соль, напротив, повышает трудоёмкость взлома, так как требует отдельного подбора для каждой записи. Для повышения защиты данных целесообразно использовать более сложные хеш-функции и длинную динамическую соль.