**МИНОБРНАУКИ РОССИИ**

**САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Факультет прикладной математики-процессов управления**

**Кафедра “фундаментальная информатика и информационные технологии”**

**ОТЧЕТ**

**по лабораторной работе №3**

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

**на тему «деобезличивание данных»**

**Вариант – 5**

**Студент гр. 22Б16-пу \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Олизько С.С.**

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

**Санкт-Петербург**

**2023 г**

Цель работы...................................................................................................................3

Задача ........................................................................................................................... 3

Теоретическая часть .....................................................................................................3

Алгоритм метода ...........................................................................................................4

Блок-схема программы .................................................................................................5

Описание программы ....................................................................................................6

Контрольный пример .....................................................................................................7

Анализ .............................................................................................................................8

Вывод ............................................................................................................................15

Источники......................................................................................................................15

**Цель работы:**

Расшифровать набор данных, зашифрованный с помощью хеш-функции с использованием модификатора входа – соли, а также проанализировать решение аналогичной задачи при различных условиях.

**Задача:**

Познакомиться с принципами работы хеш-функций и солью. Разработать программу, способную дешифровать данные, используя известные параметры: хеш-функцию, соль. Применить написанную программу к предоставленному набору данных. Проверить корректность дешифрации и правильность восстановленных данных.

**Теоритическая часть:**

Хеш-функции представляют собой алгоритмы, которые преобразуют входные данные произвольной длины в строку фиксированной длины, называемую хеш-значением или хеш-кодом. Они широко используются в криптографии и информационной безопасности для обеспечения целостности данных. Одной из ключевых характеристик хеш-функций является уникальность — отсутствие возможности получения двух различных входных значений, приводящих к одинаковому хеш-коду. Классическими примерами хеш-функций являются MD5, SHA-1 и SHA-256.

Техника взлома Brute Force предполагает перебор всех возможных комбинаций в поиске правильного пароля или ключа. В контексте расшифровки данных, Brute Force может применяться для нарушения защиты, основанной на хеш-функциях. Это медленный, но достаточно эффективный метод, особенно при отсутствии дополнительных мер безопасности.

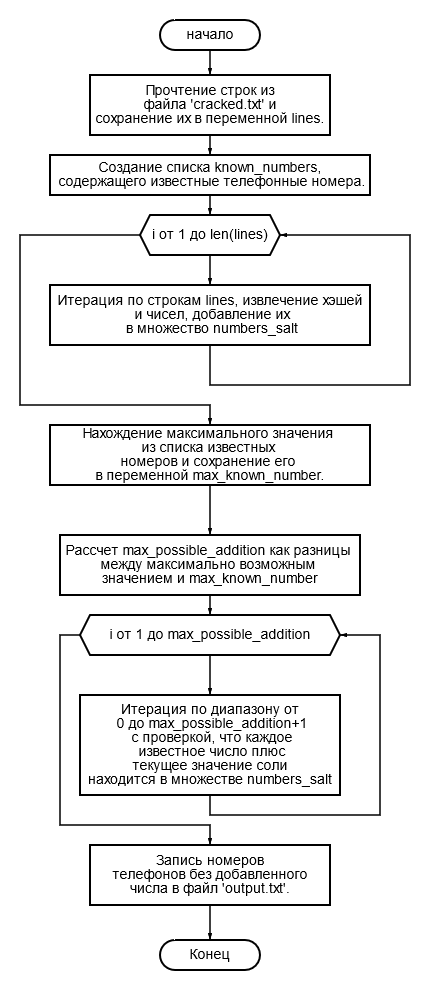
"Соль" представляет собой случайное значение, добавляемое к исходным данным перед хешированием. Использование соли устраняет проблему одинаковых хеш-значений для одинаковых входных данных, что делает атаку методом подбора Brute Force более сложной. Каждый уникальный входной набор получает свою уникальную соль, что усиливает безопасность.

При шифровании телефонных номеров с использованием хеш-функций, важно учитывать особенности хранения и обработки данных. В данной задаче, телефонные номера подвергаются хешированию с использованием соли, что увеличивает сложность атаки, основанной на анализе хеш-значений.

**Алгоритм метода:**

1. Расшифровка файла с помощью hashcat
2. Прочтение строк из файла 'cracked.txt' и сохранение их в переменной lines.
3. Создание списка known\_numbers, содержащего известные телефонные номера.
4. Итерация по строкам lines, извлечение хэшей и чисел, добавление их в множество numbers\_salt.
5. Нахождение максимального значения из списка известных номеров и сохранение его в переменной max\_known\_number.
6. Рассчет max\_possible\_addition как разницы между максимально возможным значением и max\_known\_number.
7. Итерация по диапазону от 0 до max\_possible\_addition+1 с проверкой, что каждое известное число плюс текущее значение соли находится в множестве numbers\_salt.
8. Запись номеров телефонов без добавленного числа в файл 'output.txt'.

**Блок схема программы:**



*Рис 2: блок-схема*

**Описание программы:**

Программная реализация написана на языке python 3.10.4. В программе использовались 8 структур данных.

Описание структур данных:

*Таблица 1 : структуры данных*

|  |  |  |  |
| --- | --- | --- | --- |
| Имя структуры | Тип структуры | Тип хранимых данных | Описание |
| lines | List | String | Список строк, содержащих данные, считанные из файла 'cracked.txt'. |
| known\_numbers | List | Integer | Список известных телефонных номеров. |
| numbers\_salt | Set | Integer | Множество, предназначенное для хранения хешей и соответствующих чисел из файла 'cracked.txt'. |
| max\_known\_number | Integer | Integer | Максимальное известное значение из списка известных номеров. |
| max\_possible\_addition | Integer | Integer | Максимальное значение, которое может быть добавлено к числу, чтобы не изменить количество разрядов. |
| global\_salt | Integer | Integer | Глобальное значение соли, которое будет добавлено к каждому номеру. |
| result\_numbers | List | Integer | Список номеров телефонов после вычитания глобальной соли. |
| output\_file | File | Integer | Файл 'output.txt' для записи номеров телефонов без добавленного числа. |

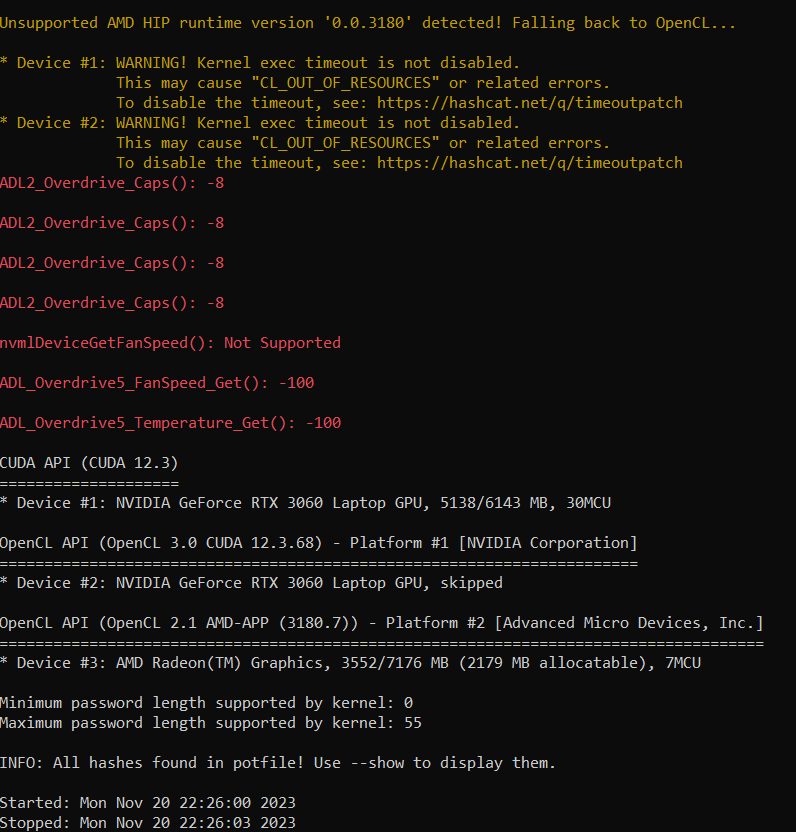
**Рекомендации для пользователя:**

1. Необходимо запустить cmd (Win+R -> cmd)
2. Выполнить команду hashcat -m 0 -a 3 -O -o <абсолютный путь до выходного файла (та же директория где находится программа .py> <абсолютный путь до входного файла> -1 89 ?1?d?d?d?d?d?d?d?d?d?d
3. Запустить .py файл

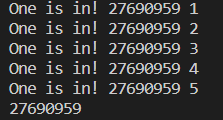
Необходимо установить программу hashcat. Необходимо проверить наличие всех необходимых драйверов для программы hashcat (в частности Nvidia CUDA Toolkit).

**Исходный код программы доступен по ссылке ниже:**

<https://github.com/StephanOlizko?tab=repositories>

**Контрольный пример:**  
  
****

*Рис 5: пример окна программы*



*Рис 6: пример вывода программы*

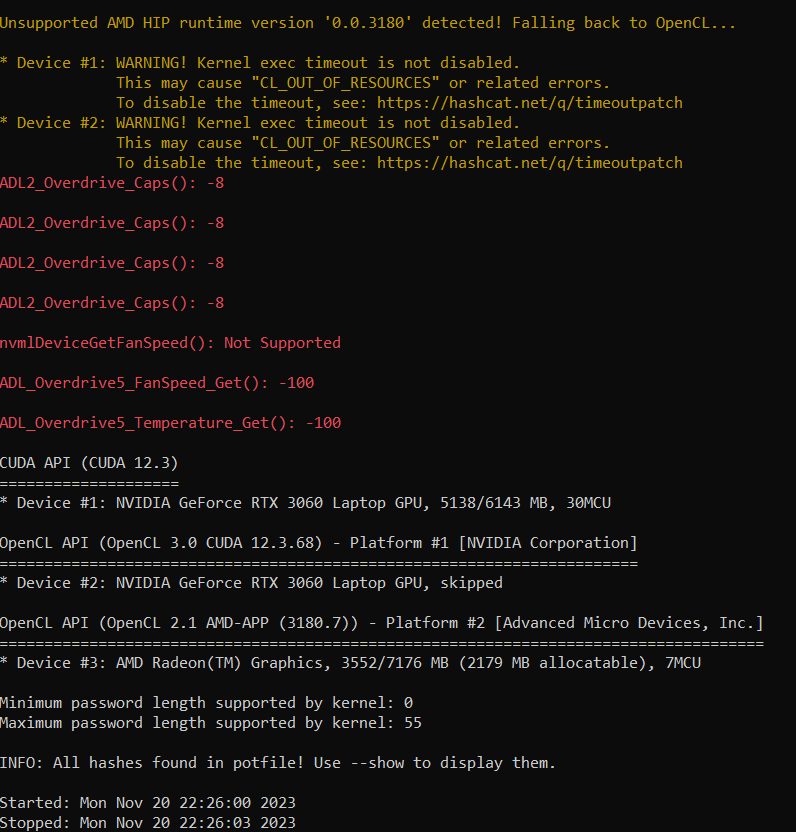
**Анализ результатов работы алгоритма и вводных условий:**

**MD5 (Message Digest Algorithm 5)**

MD5 является одним из самых распространенных алгоритмов хэширования. Он производит фиксированную 128-битную хэш-сумму (32 символа шестнадцатеричного кода) для входных данных произвольной длины.

* Используется в проверке целостности файлов.
* Распространено в хранилищах паролей (хотя не рекомендуется из-за уязвимостей).

MD5 считается устаревшим и небезопасным, так как существуют методы коллизий (возможность двух различных данных, дающих одинаковый хэш).

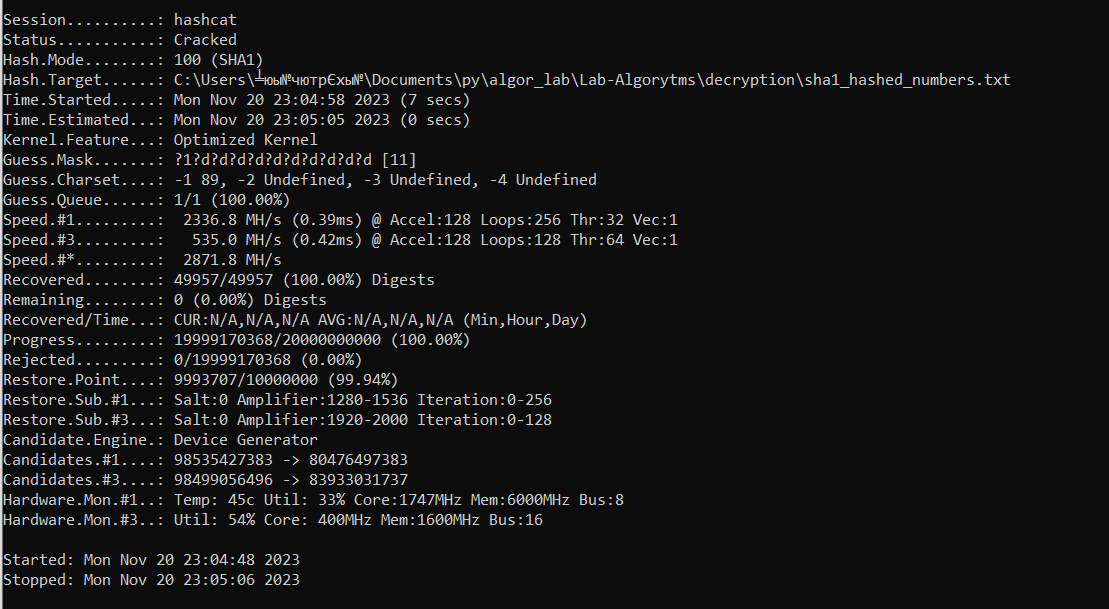
****

**SHA-1 (Secure Hash Algorithm 1)**

SHA-1 является алгоритмом хэширования, созданным для замены MD5. Он производит фиксированную 160-битную хэш-сумму (40 символов шестнадцатеричного кода) для входных данных произвольной длины.

* Использовался в безопасности и цифровой подписи.
* Однако, считается устаревшим и подверженным коллизиям.

SHA-1 также считается небезопасным из-за возможности коллизий.

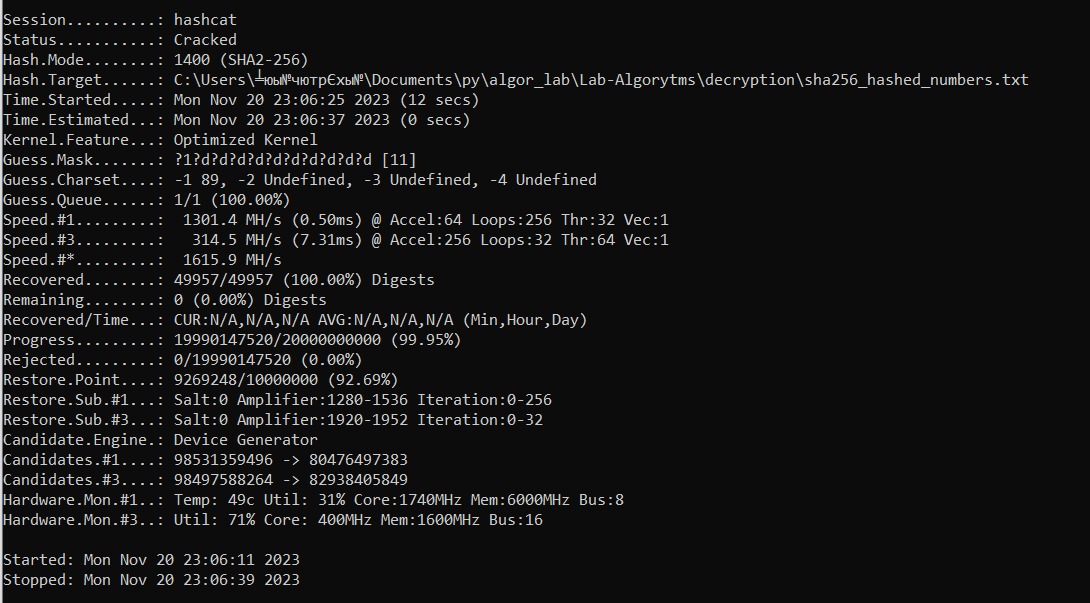
****

**SHA-256 (Secure Hash Algorithm 256-bit)**

SHA-256 принадлежит семейству SHA-2 и производит 256-битную хэш-сумму (64 символа шестнадцатеричного кода).

* Широко используется в криптографических приложениях, блокчейнах, цифровой подписи.

Более безопасный по сравнению с MD5 и SHA-1.

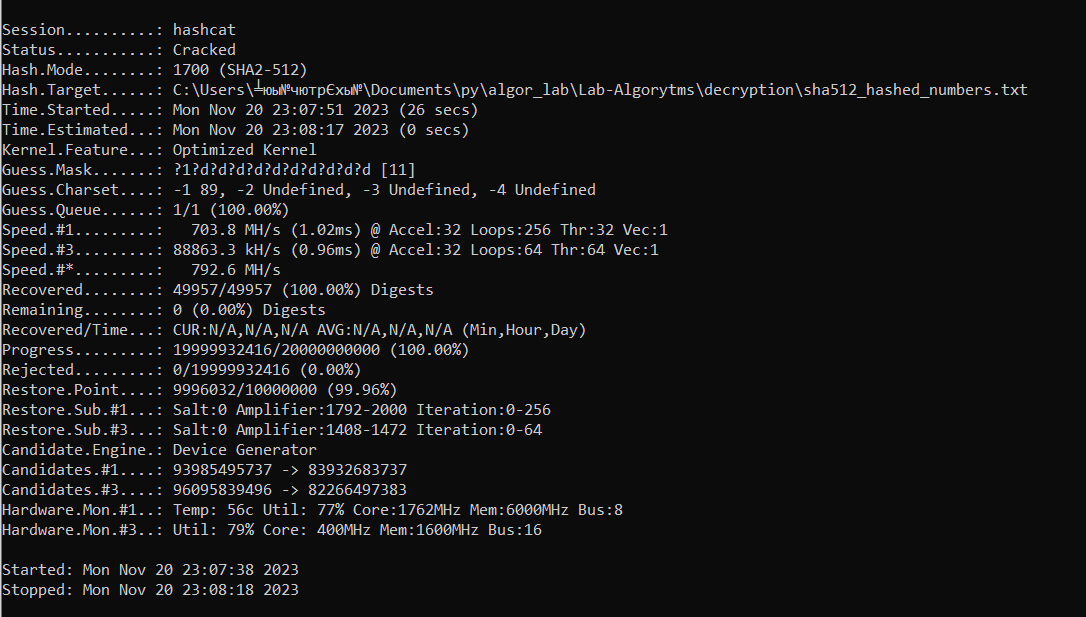
****

**SHA-512 (Secure Hash Algorithm 512-bit)**

SHA-512 также принадлежит семейству SHA-2 и генерирует 512-битную хэш-сумму (128 символов шестнадцатеричного кода).

* Используется в высокоуровневых криптографических приложениях и технологиях безопасности.

Более безопасный и обладает бОльшей длиной хэша по сравнению с SHA-256.

****

**Анализ:**

Чем длиннее хэш, тем сложнее взломать методом подбора.

MD5 и SHA-1 больше не рекомендуются для криптографического использования из-за выявленных уязвимостей. SHA-256 и SHA-512 считаются более безопасными на текущий момент.

**Анализ разных видов модификаторов входа (соли):**

1. **Численная соль:**

Численная соль представляет собой дополнительное числовое значение, добавляемое к паролю перед хэшированием.

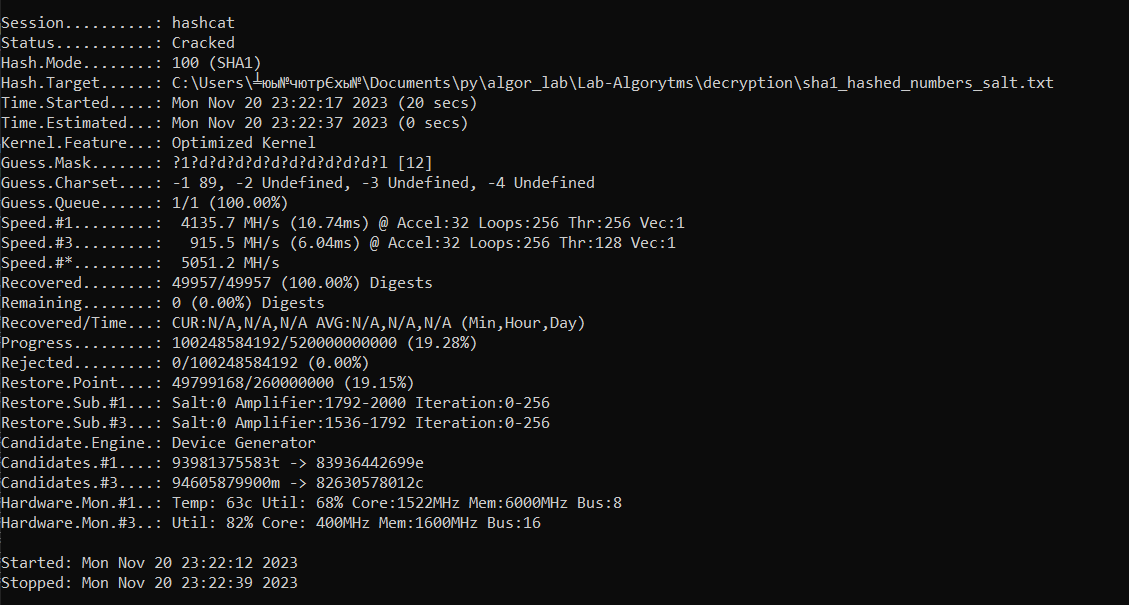
Обычно применяется в сценариях, где требуется дополнительная уникальность, но без увеличения сложности расшифровки.

**2. Буквенная соль:**

Буквенная соль представляет собой строку символов, добавляемую к паролю перед хэшированием.

Увеличивает время расшифровки, поскольку требуется проводить атаки перебора для каждой добавочной буквенной соли.

Обеспечивает дополнительную уникальность и усложняет атаки с использованием заранее вычисленных таблиц (таблиц радужных хэшей).

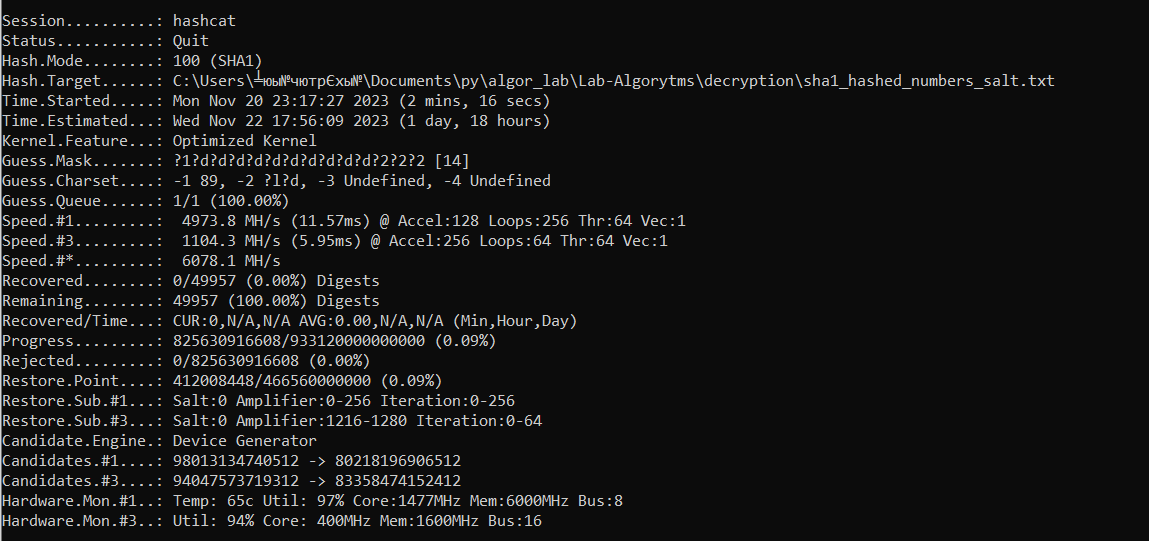
****

**3. Комбинированная соль:**

Комбинированная соль объединяет численные и буквенные элементы в единое добавочное значение.

Увеличивает время расшифровки еще больше, причем нелинейно. Каждая новая буква или цифра увеличивает сложность взлома, так как требуется атака для каждой уникальной комбинации.

Максимально повышает уровень уникальности и безопасности хэшей, но также увеличивает вычислительную сложность.

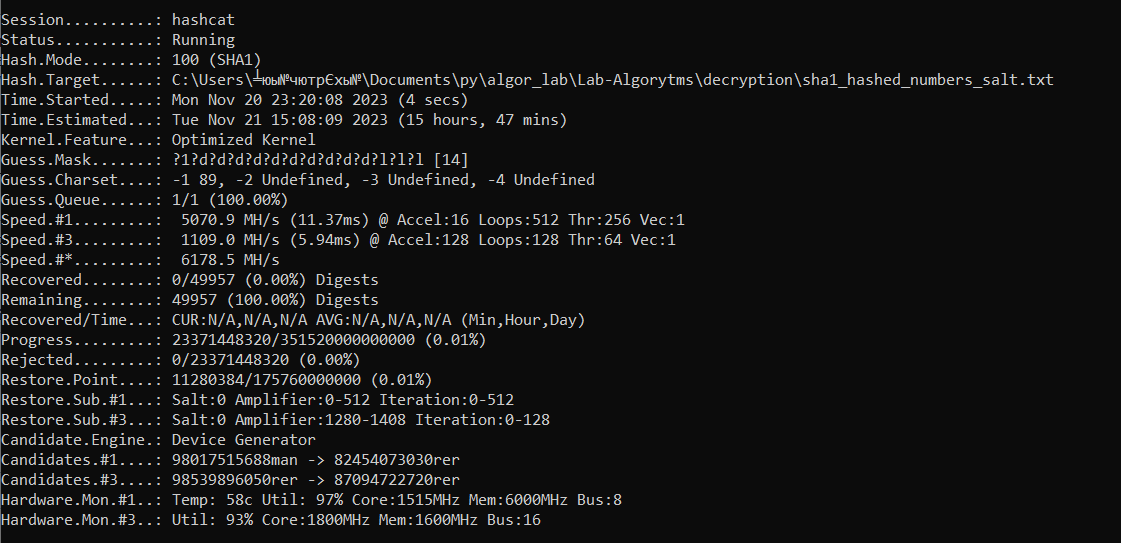
****

**4. Зависимость времени от количества буквенной соли:**

Увеличение количества буквенной соли приводит к экспоненциальному увеличению времени расшифровки.

Каждый символ в буквенной соли увеличивает количество возможных комбинаций, что заметно замедляет атаки перебора.

Даже небольшое количество буквенной соли может существенно повысить безопасность, но добавление слишком большого количества может привести к увеличению нагрузки на систему.

****

*Таблица 2 : время дешифровки*

|  |  |  |
| --- | --- | --- |
| *Алгоритм шифрования* | *Вид Соли* | *Время дешифровки* |
| MD5 | Численный | 3 сек |
| SHA1 | Численный | 18 сек |
| SHA256 | Численный | 28 сек |
| SHA512 | Численный | 40 сек |
| SHA1 | Буквенный (1 символ) | 20 сек |
| SHA1 | Буквенный (3 символа) | 15 ч 47 мин |
| SHA1 | Комбинированный | 42 ч |

**Вывод:**

Расшифрован набор данных, зашифрованный с помощью хеш-функции. Процесс расшифровки требует знания использованной соли для правильного вычисления хэша. Было рассмотрено использование различных алгоритмов хеширования (MD5, SHA-1, SHA-256, SHA-512) и разные виды соли (численная, буквенная).

Буквенная соль дополнительно усложняет атаки перебора, но также увеличивает время расшифровки. Комбинированная соль создает наивысший уровень уникальности, но требует больше вычислительных ресурсов.

Важно находить баланс между безопасностью и производительностью при выборе вида и объема соли.

**Источники:**

1. <https://hashcat.net/wiki/doku.php?id=mask_attack>
2. <https://hashcat.net/wiki/doku.php?id=mask_attack>
3. <https://docs.python.org/3/hashlib>