**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 – 5**

**Студент гр. 23Б16-пу**

**Горыцнцев Р.Н.**

**Преподаватель**

**Дик А.Г.**

**Санкт-Петербург**

**2024 г.**

Оглавление

1. [Цель работы 3](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013329)
2. [Описание задачи (формализация задачи) 3](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013330)
3. [Теоретическая часть 4](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013331)
4. [Основные шаги программы 5](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013332)
5. [Блок схема программы 6](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013333)
6. [Описание программы 7](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013334)
7. [Рекомендации пользователя 8](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013335)
8. [Контрольный пример 9](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013338)
9. [Анализ результатов работы 10](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013339)
10. [Вывод 17](file:///C:\Users\Romchik\Downloads\Telegram%20Desktop\Отчет.docx#_Toc180013340)

# **Цель работы**

Цель данной лабораторной работы — расшифровать набор данных, зашифрованный с помощью хеш-функции с использованием модификатора входа – соли, а также проанализировать решение аналогичной задачи при различных условиях.

# **Описание задачи (формализация задачи)**

1. Изучить особенности шифрования телефонных номеров:

Понять, как телефонные номера шифруются с использованием хеш-функций и соли, и как соль влияет на стойкость шифрования. Ознакомиться с различными типами хеш-функций и их свойствами.

1. Написать программу для дешифрования датасета:

Разработать программу, которая будет пытаться дешифровать датасет, зашифрованный с использованием хеш-функции и соли. Программа должна учитывать различные типы и длины соли, а также поддерживать несколько хеш-функций.

1. Протестировать программу на выданном варианте:

Проанализировать время работы программы и количество успешно дешифрованных номеров.

1. Протестировать с еще минимум 3-мя различными хеш-функциями:

Исследовать влияние различных хеш-функций на скорость дешифрования.

1. Исследовать влияние различных факторов на скорость расшифровки:

Определить, как тип и длина соли, а также выбор хеш-функции влияют на скорость расшифровки датасета. Ответить на вопрос – сколько нужно знать телефонов из предложенного датасета, чтобы гарантированно его взломать?

# **Теоретическая 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 SHA-1, SHA-256 и SHA-512.

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. В данной лабораторной работе Brute Force будет использоваться для перебора всех возможных телефонных номеров и сравнения их хеш-значений с данными в датасете.

"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.

Существует множество хеш-функций, каждая со своими особенностями и сферами применения. В рамках данной лабораторной работы рассматриваются следующие типы:

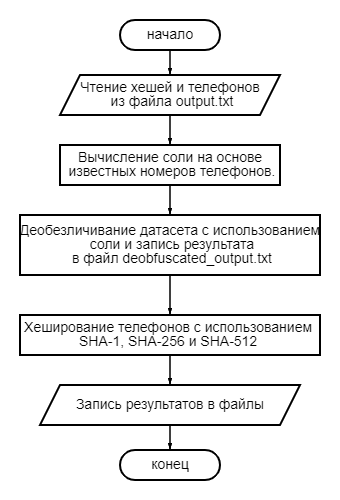
* SHA-1: Более безопасная альтернатива MD5, но также подвержена коллизиям.
* SHA-256: Современная функция, обеспечивающая высокий уровень безопасности.
* SHA-512: Обеспечивает высокий уровень безопасности и имеет большую длину хеш-значения.

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.

# **Основные шаги программы**

* 1. Расшифровка файла с помощью hashcat и сохранение в файл output.txt
  2. Чтение хешей и номеров телефонов из файла output.txt.
  3. Вычисление соли на основе известных телефонных номеров и хешей.
  4. Деобезличивание датасета с использованием вычисленной соли и запись результатов в файл deobfuscated\_output.txt.
  5. Извлечение номеров телефонов из файла deobfuscated\_output.txt.
  6. Хеширование извлеченных номеров телефонов с использованием SHA-1, SHA-256 и SHA-512 и запись результатов в файлы sha1\_hashes.txt, sha256\_hashes.txt, и sha512\_hashes.txt.

# **Блок схема программы**



*Рис 1. Блок схема программы*

# **Описание программы**

Программная реализация написана на языке Python 3.12 с использованием следующих библиотек: hashlib. Программа организована через единый файл с акцентом на деобезличивание данных, зашифрованных с использованием хеш-функций и соли. В процессе разработки программы использовались 7 функций, каждая из которых имеет чётко определённое назначение:

*Таблица 1. lab\_3.py*

|  |  |  |
| --- | --- | --- |
| Функция | Описание | Возвращаемое значение |
| read\_hashes\_and\_phones | Считывает хеши и номера телефонов из файла output.txt. | str |
| compute\_salt | Вычисляет соль на основе известных телефонных номеров и хешей. | None |
| deobfuscate\_dataset | Использует вычисленную соль для восстановления оригинальных номеров телефонов и записывает результаты в файл deobfuscated\_output.txt. | None |
| extract\_phones | Извлекает номера телефонов из файла deobfuscated\_output.txt. | str |
| hash\_phones | Хеширует извлеченные номера телефонов с использованием SHA-1, SHA-256 и SHA-512 и записывает результаты в файлы sha1\_hashes.txt, sha256\_hashes.txt, и sha512\_hashes.txt. | None |
| main | Основная функция, которая вызывает все остальные функции в правильном порядке для выполнения задачи деобезличивания и хеширования. | None |
| read\_hashes\_and\_phones | Считывает хеши и номера телефонов из файла output.txt. | Список строк |

# **Рекомендации пользователя**

1. Убедитесь, что у вас установлен Python 3.12 и библиотека hashlib.
2. Убедитесь, что Hashcat установлен и добавлен в системную переменную PATH, чтобы его можно было запускать из командной строки.
3. Откройте командную строку (Win+R -> cmd).
4. Выполнить команду: hashcat -m 0 -a 3 <входной файл> ?d?d?d?d?d?d?d?d?d?d?d --force -D 1,2 -o <выходной файл>

# **Контрольный пример**

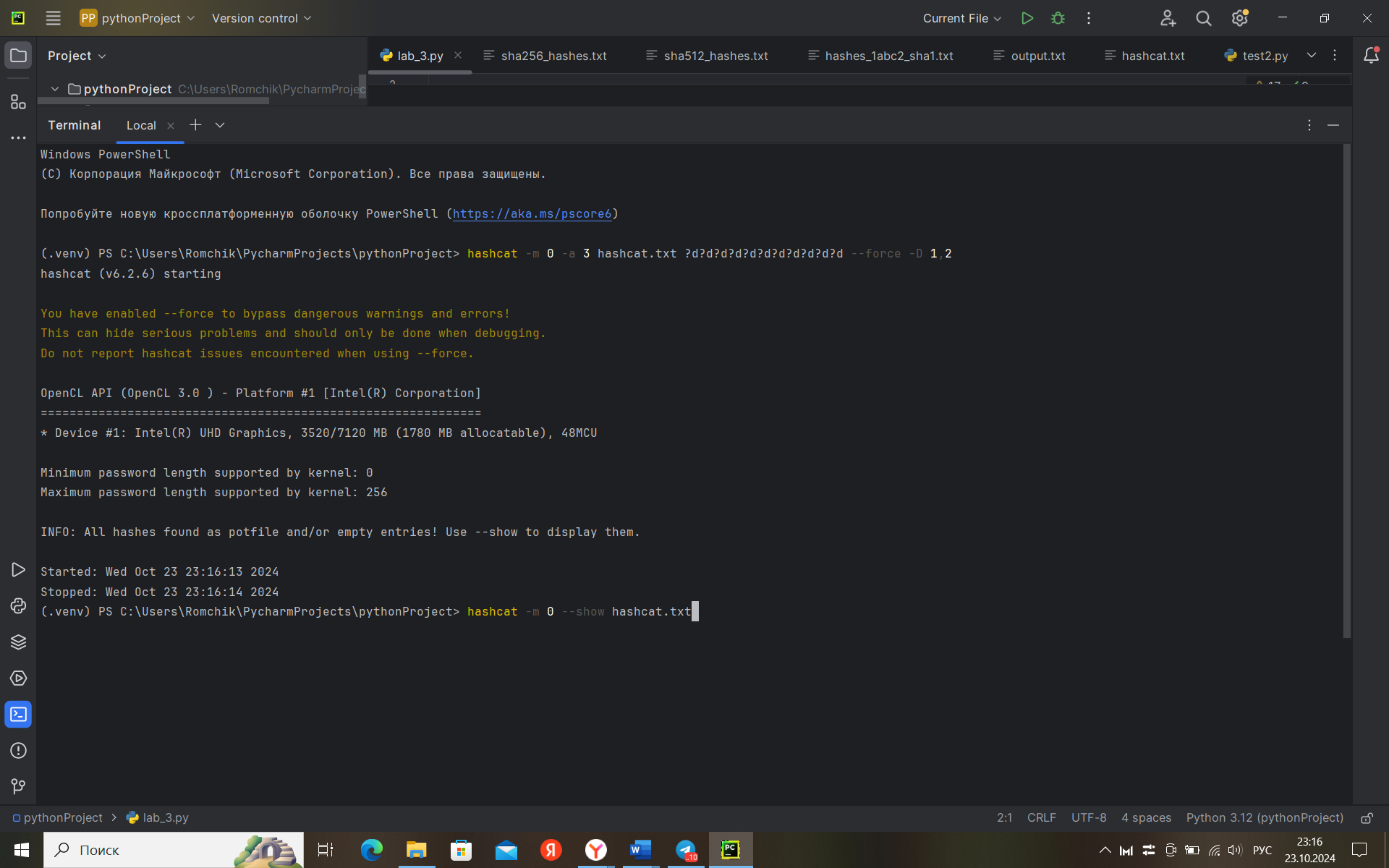


Рис 2. Пример расшифровки хешей с помощью hashcat

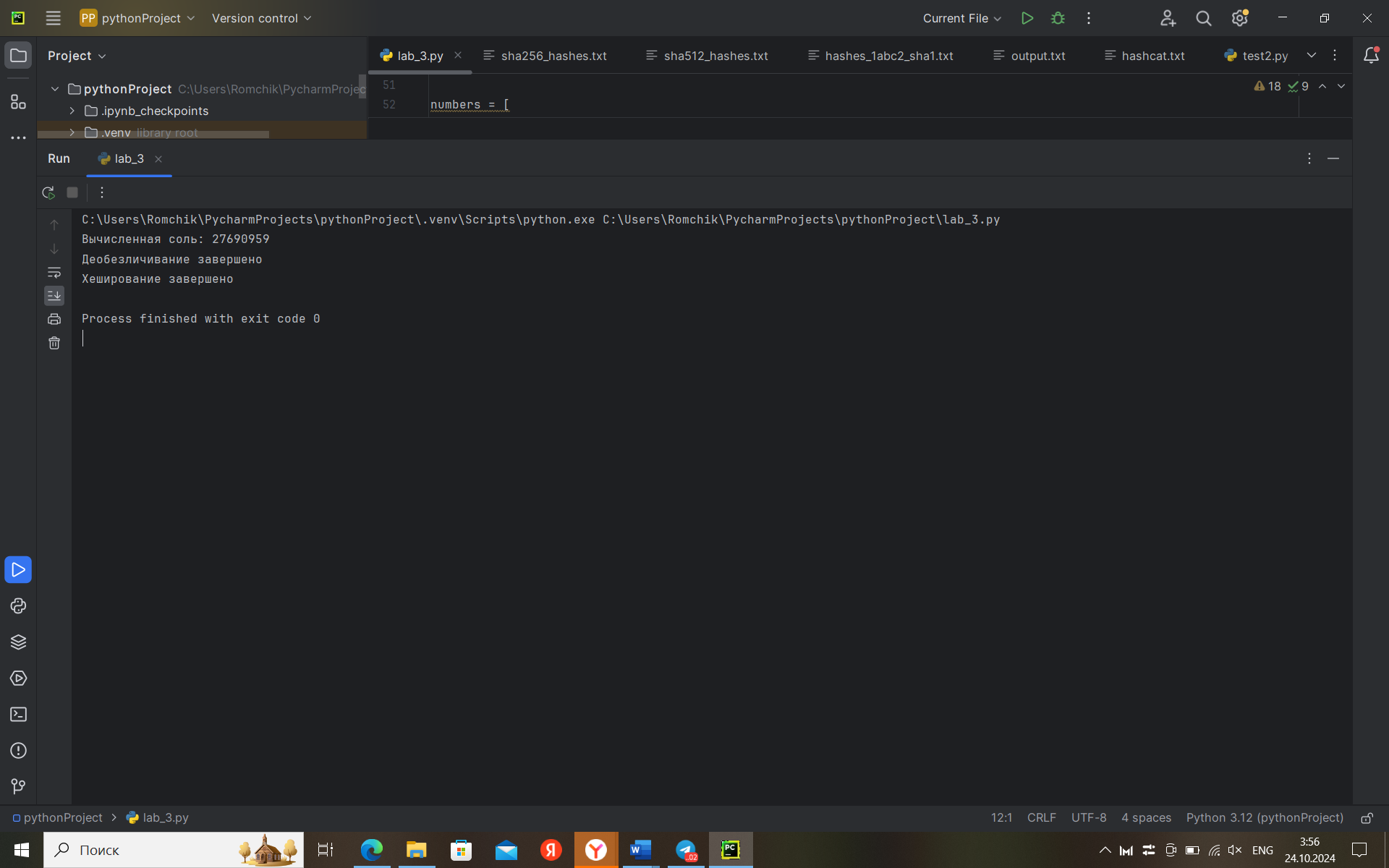


Рис 3. Пример вывода программы

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

**SHA-1** является одним из алгоритмов хэширования, производящих фиксированную 160-битную хэш-сумму (40 символов шестнадцатеричного кода) для входных данных произвольной длины.  
• Используется в проверке целостности файлов и сообщений.  
• Ранее широко применялся в хранилищах паролей, но из-за уязвимостей к коллизиям теперь не рекомендуется.  
SHA-1 считается устаревшим и небезопасным, так как существуют методы коллизий (возможность двух различных данных, дающих одинаковый хэш).

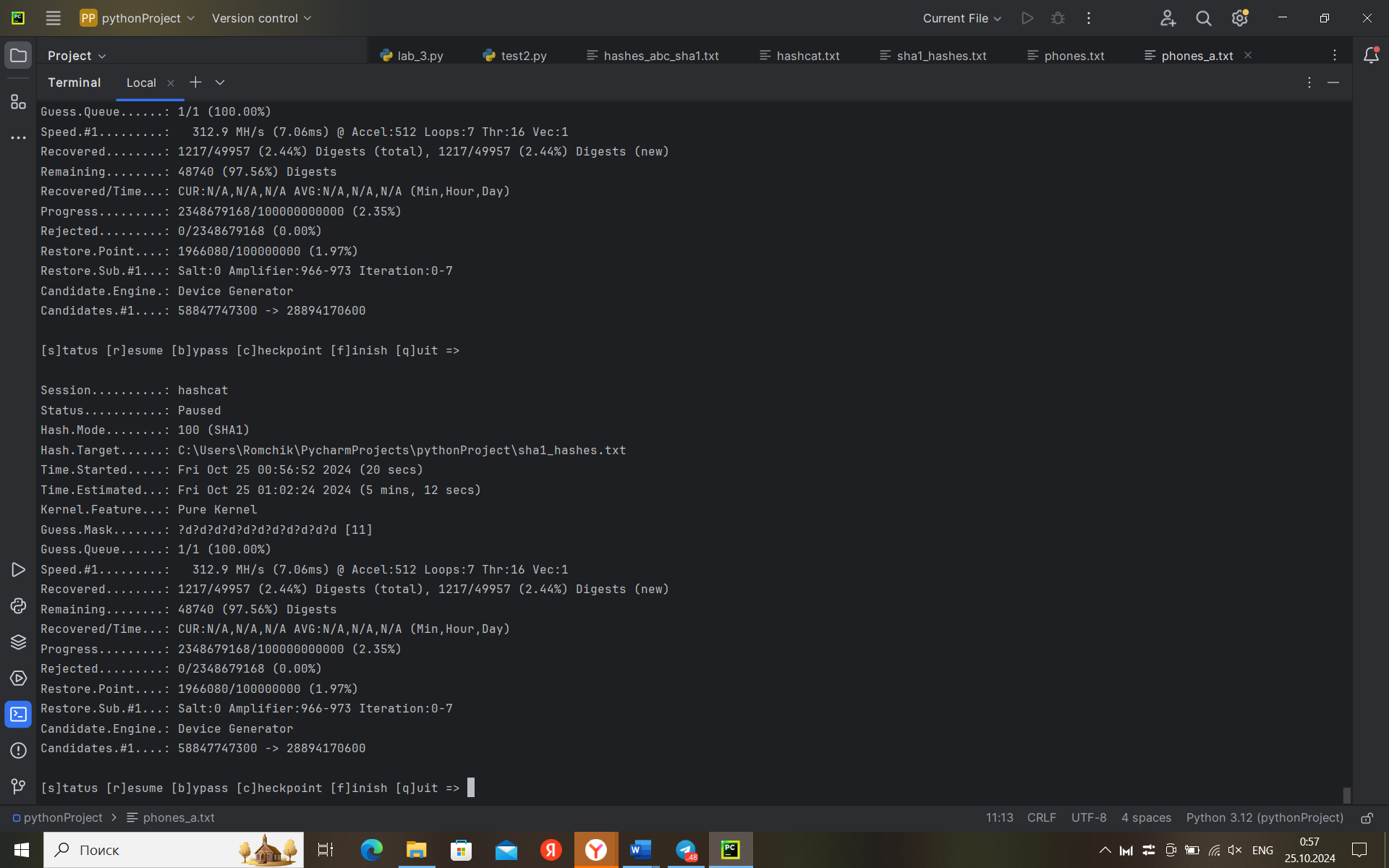


Рис 4. SHA-1 без соли

**SHA-256** является одним из алгоритмов хэширования, производящих фиксированную 256-битную хэш-сумму (64 символа шестнадцатеричного кода) для входных данных произвольной длины.  
• Широко используется в криптовалютах, таких как Bitcoin.  
• Применяется для проверки целостности программного обеспечения.  
• Может использоваться для хранения паролей с применением соли и медленного алгоритма хеширования.  
SHA-256 считается безопасным и рекомендуется для большинства современных задач.

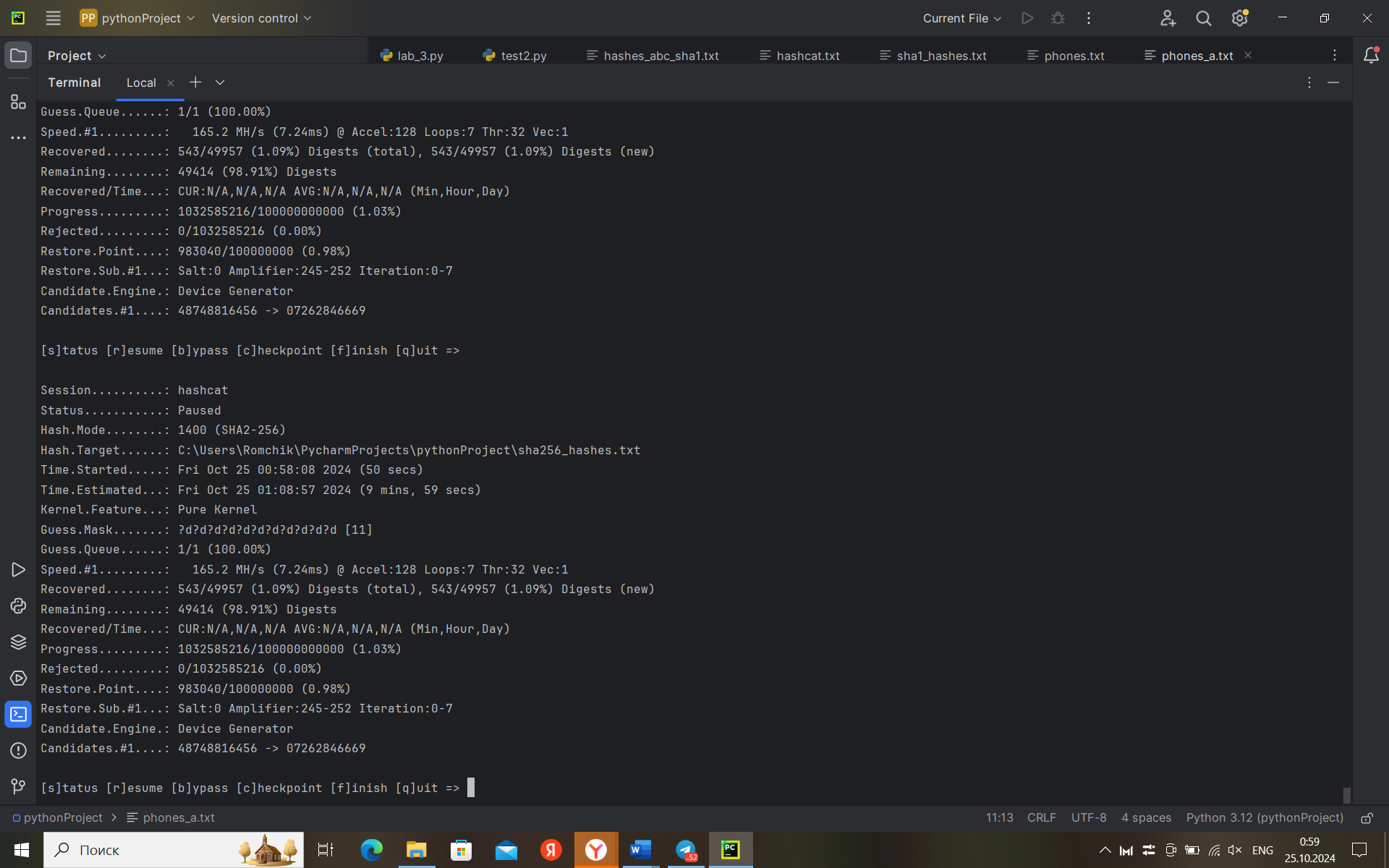


Рис 5. SHA-256 без соли

**SHA-512** является одним из алгоритмов хэширования, производящих фиксированную 512-битную хэш-сумму (128 символов шестнадцатеричного кода) для входных данных произвольной длины.  
• Широко используется в криптовалютах, таких как Ethereum.  
• Применяется для хранения паролей с применением соли и медленного алгоритма хеширования.  
• Используется для аутентификации сообщений.  
SHA-512 считается еще более безопасным, чем SHA-256, и рекомендуется для задач, требующих максимальной защиты.

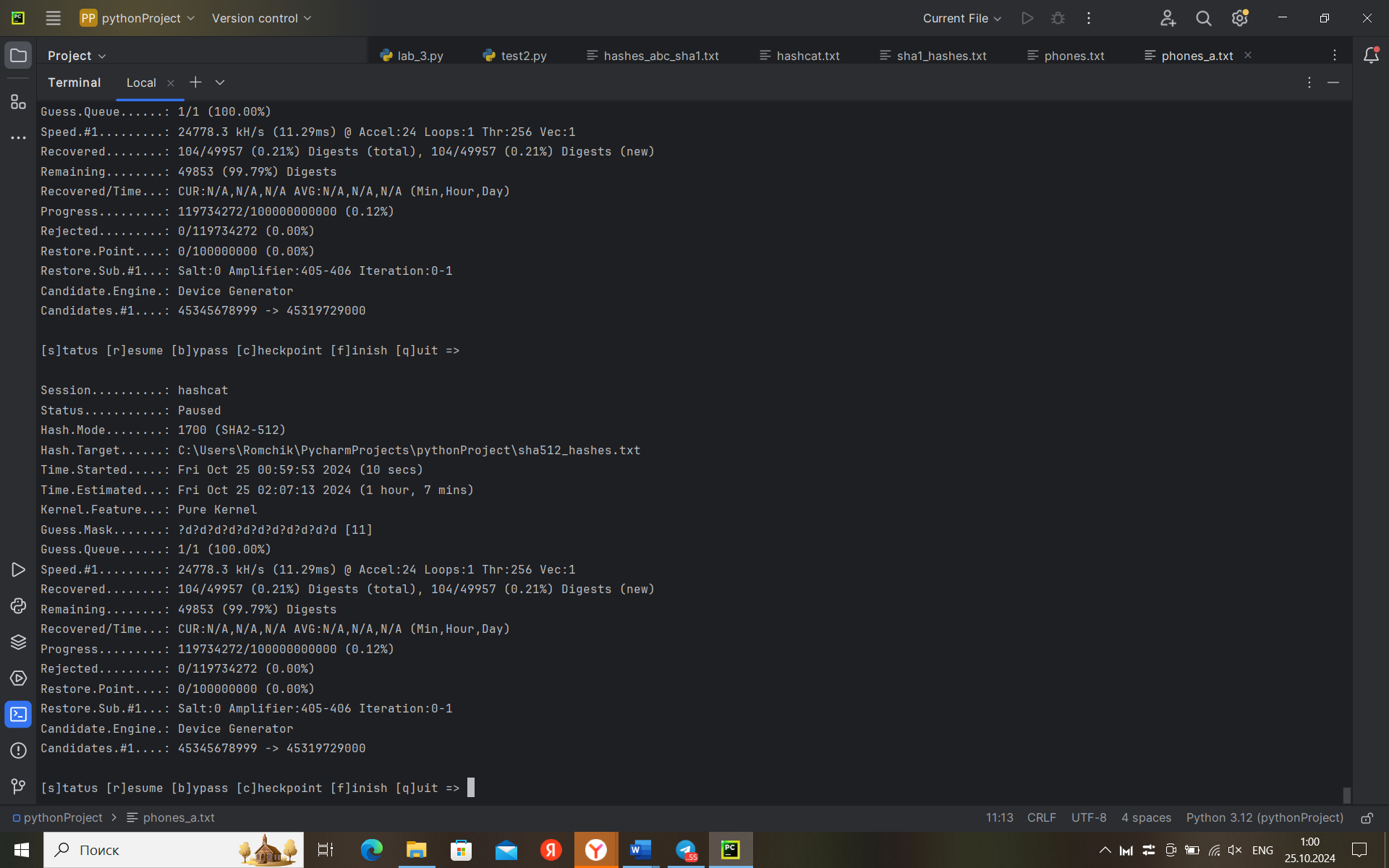


Рис 6. SHA-512 без соли

**Выводы:**

* Чем длиннее хэш, тем больше времени требуется для перебора всех возможных значений. Это замедляет Brute force атаки, делая их менее эффективными.
* Более безопасные хэш-функции, такие как SHA-256 и SHA-512, замедляют процесс расшифровки, что делает их более эффективными для защиты паролей и других чувствительных данных

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

**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.

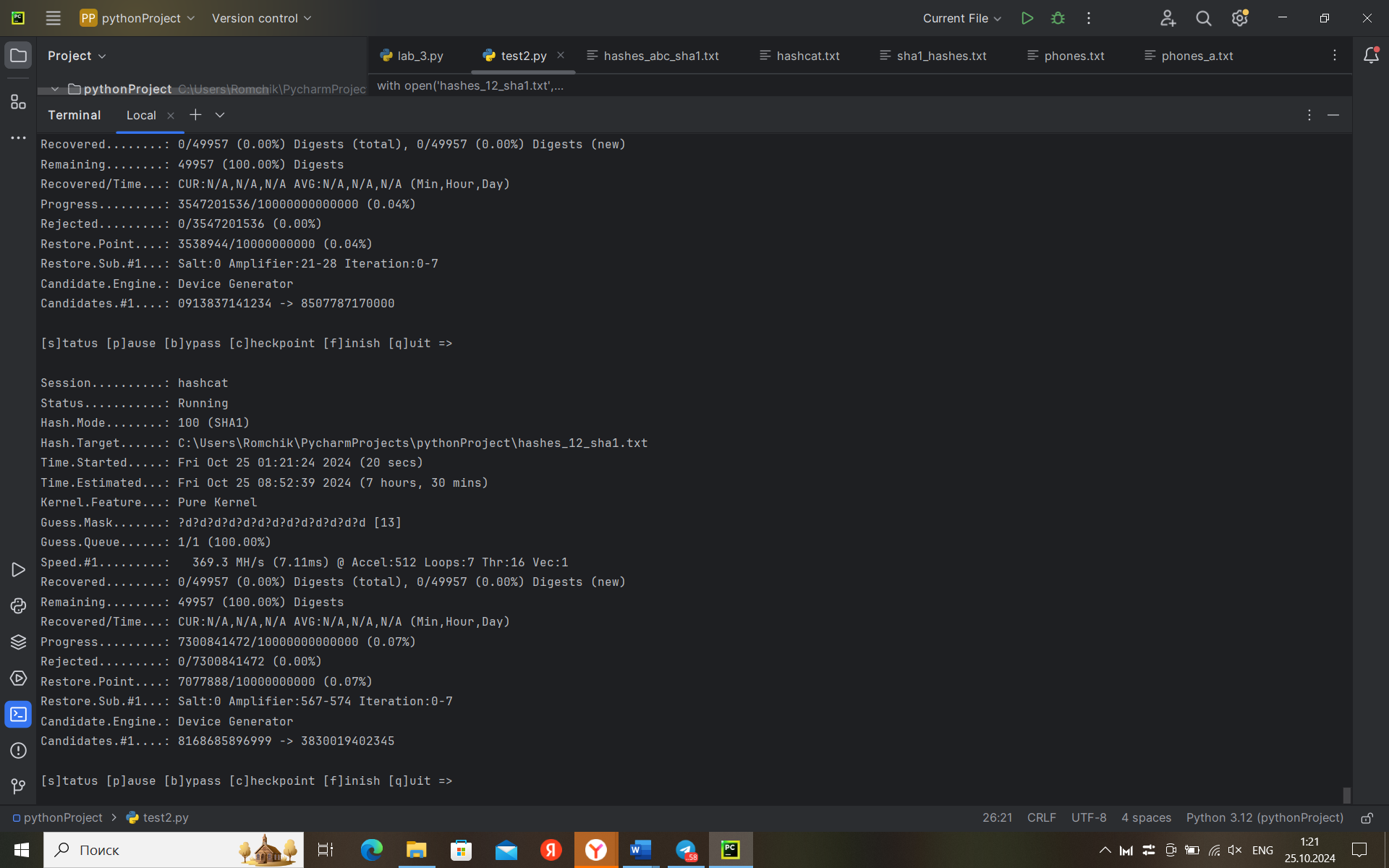


Рис 7. Пример численной соли

**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).

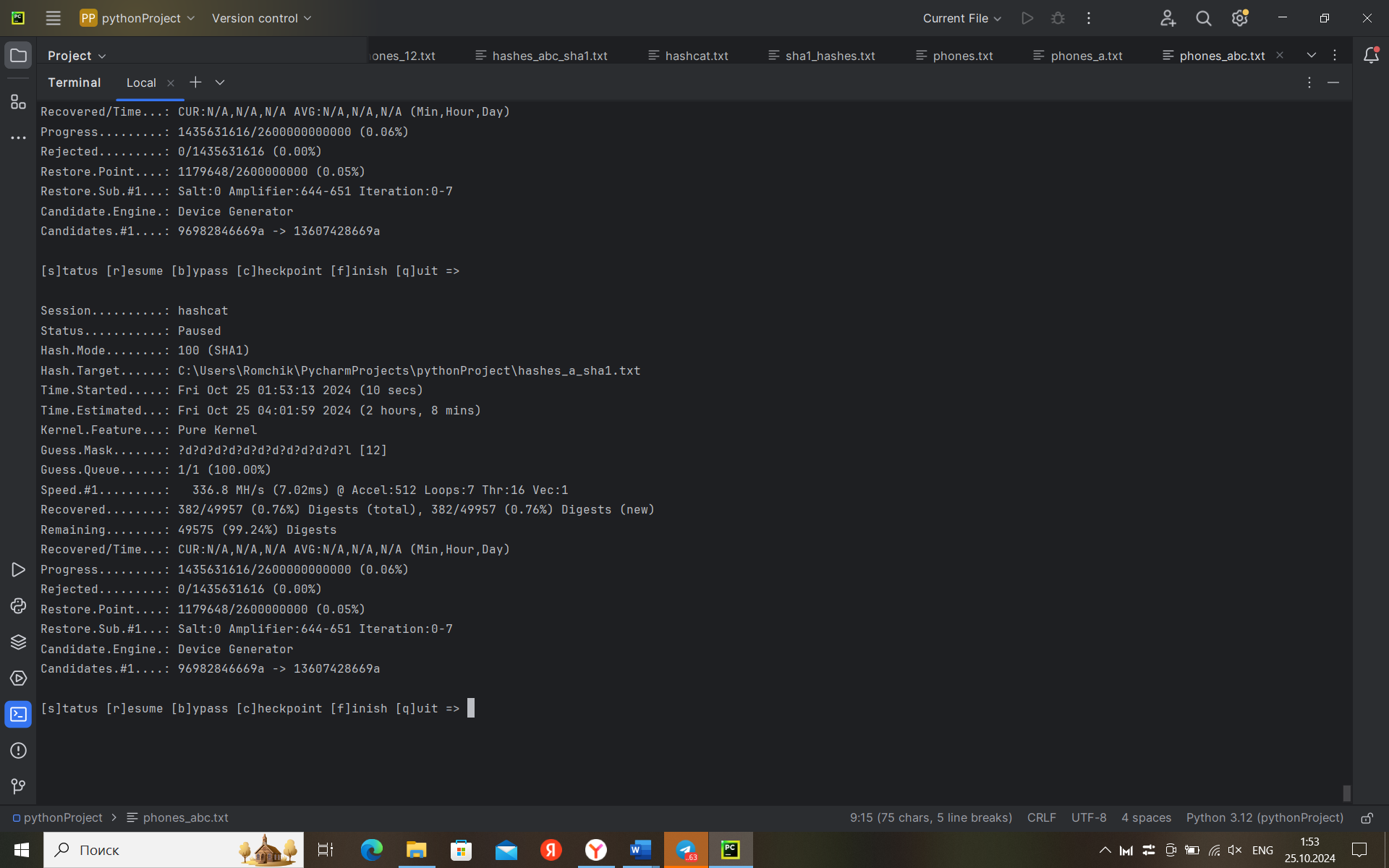


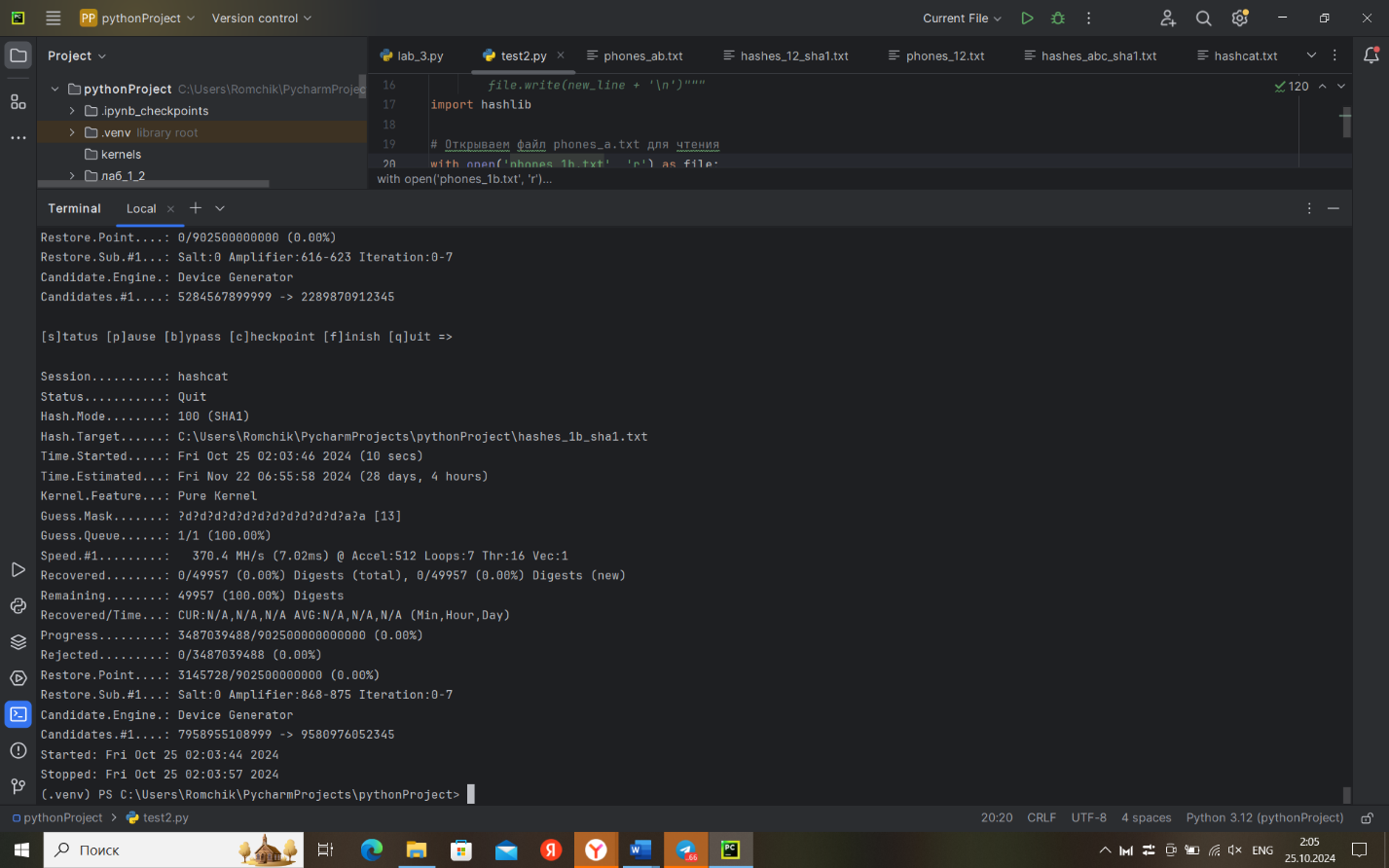
Рис 8. Пример буквенной соли (1 символ)

**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.

  
Рис 9. Пример комбинированной соли

**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.

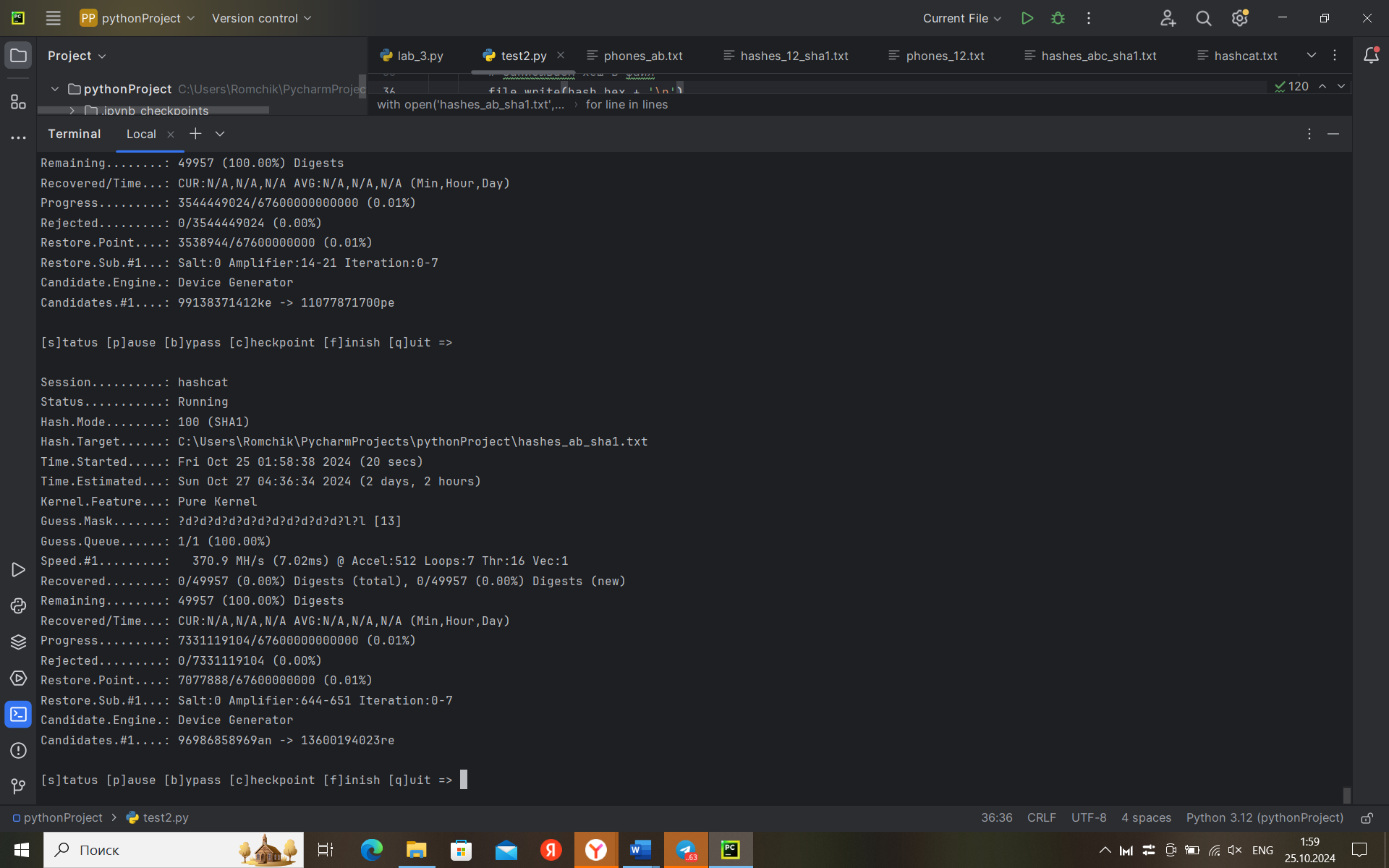


Рис 10. Пример буквенной соли (2 символа)

**Вывод:**

В ходе лабораторной работы было установлено, что использование соли значительно повышает безопасность хеширования, но приводит к значительному увеличению времени расшифровки. Длина соли и тип хеш-функции также оказывают существенное влияние на скорость расшифровки: чем длиннее соль и чем более современная и безопасная хеш-функция используется, тем больше вычислительных ресурсов требуется для взлома. Таким образом, использование соли и современных хеш-функций является эффективным способом защиты данных, несмотря на увеличение времени расшифровки.