SIBINTEK CTF 2025

Задания

09.11.2025

Название: Chess_incoming

Категория: Stego

Очки: динамическое начисление

Описание: Сотрудники ДНК Сибирь часто играют в шахматы в свободное время. Однако в последние несколько дней отдел информационной безопасности заподозрил что-то странное - сотрудники зачем-то приводят партии к непонятным позициям, после чего фотографируют результат партии.

Отделу ИБ удалось перехватить один из таких файлов. Разберитесь в чем суть.

Флаг: Sibintek{1nc0m1ng_c4ll_f0r_ch3ss}

01

Первичный анализ файла task.jpg.
Начнем с анализа предоставленного изображения шахматной доски. Проверим, не является ли файл полиглотом (файлом, который одновременно является несколькими форматами).

Используем команду `binwalk` для анализа структуры файла:

```
binwalk task.jpg
```

Результат показывает, что внутри JPG файла находится ZIP-архив. Извлечем ZIP-архив из изображения:

```
binwalk -e task.jpg
# или
dd if=task.jpg of=archive.zip bs=1 skip=<offset>
```

Попытаемся открыть извлеченный архив:

```
unzip task.zip
```

Архив защищен паролем.
Необходимо найти способ получить пароль.

02

Извлечение пароля из шахматной позиции.
Внимательно изучим шахматную доску. Позиция выглядит необычной и явно не из реальной партии. Возможно, расположение фигур скрывает информацию.

Попробуем рассмотреть шахматную доску как матрицу для кодирования данных:
- Белая фигура = 1
- Черная фигура = 0
- Пустая клетка = разделитель (переход к следующему числу)

Считываем доску построчно сверху вниз (от 8-й горизонтали к 1-й):

Горизонталь 8:
- a8: белая ладья = 1
- b8: черный конь = 0
- c8: черный слон = 0
- d8: белый конь = 1
- e8: пустая
- f8: пустая
- g8: белый слон = 1
- h8: черная ладья = 0

Получаем: `1001` (пустая) (пустая) `10` → числа: 9, 2

Горизонталь 7:
- a7: пустая
- b7: пустая
- c7: пустая
- d7: белый конь = 1
- e7: черный король = 0
- f7: белая пешка = 1
- g7: пустая
- h7: пустая

Получаем: (пустая)(пустая)(пустая) `101` → число: 5

Горизонталь 6:
- a6: белая пешка = 1
- b6: черная пешка = 0
- c6: белая пешка = 1
- d6: черная пешка = 0
- e6: черный конь = 0
- f6-h6: пустые

Получаем: `10100` → число: 20

Горизонталь 5:
- Пустая, пустая, белый слон = 1, черная ладья = 0, черная пешка = 0, черная пешка = 0, белая пешка = 1

Получаем: (пустая)(пустая) `10001` → число: 17

Горизонталь 4:
- Все пустые кроме: f4 белая ладья = 1, g4 черная пешка = 0, h4 черная пешка = 0

Получаем: `100` → число: 4

Горизонталь 3:
- a3: белый ферзь = 1
- b3: черный ферзь = 0
- c3: черная пешка = 0
- d3: черная пешка = 0
- e3: белая пешка = 1
- f3: черный слон = 0
- остальные пустые

Получаем: `100010` → число: 34

Горизонталь 2:
- Пустые клетки до g2: белая пешка = 1, h2: белая пешка = 1

Получаем: `11` → число: 3

Горизонталь 1:
- Пустые, пустые, пустые, d1: белая пешка = 1, e1: белый король = 1
- остальные пустые

Получаем: `11` → число: 3

Собираем все извлеченные числа: 9, 2, 5, 20, 17, 4, 34, 3, 3

Удаляем пробелы и получаем пароль: `925201743433`

03

Открытие архива.
Используем полученный пароль для распаковки архива:

```
unzip -P 925201743433 task.zip
```

Внутри находим:
- `moves.txt` - текстовый файл с подсказкой
- `test.zip` - еще один защищенный паролем архив

Изучим содержимое `moves.txt`:

```
Сегодня задействуем коня. Как некогда гордый муж, рожденный
в Мегалополисе, ведущий свое войско вперед, а также тот кто
ведал в истории. Ты знаешь, что делать
h3-g5
g5-f7
f7-d6
d6-f5
f5-d4
d4-e6
e6-f4
f4-h5
h5-f6
f6-d7
```

Подсказка содержит намек на исторического персонажа:
- "Гордый муж, рожденный в Мегалополисе"
- "Тот кто ведал в истории"

Это описание древнегреческого историка **Полибия** (около 200-118 до н.э.), который родился в Мегалополисе и известен своей системой шифрования - **Квадратом Полибия**.

Квадрат Полибия - это таблица 5×5 (или в нашем случае можно использовать шахматную доску 8×8), где каждая буква алфавита соответствует координатам на доске.

Используя шахматную доску как квадрат Полибия, где:
- Столбцы: a, b, c, d, e, f, g, h
- Строки: 1, 2, 3, 4, 5, 6, 7, 8

Составим соответствие букв латинского алфавита координатам (26 букв, используем часть доски):

```
   a  b  c  d  e  f  g  h
8  A  B  C  D  E  F  G  H
7  I  J  K  L  M  N  O  P
6  Q  R  S  T  U  V  W  X
5  Y  Z  .  .  .  .  .  .
```

Из подсказки: "Сегодня задействуем коня" - значит используем ходы коня по шахматной доске.

Преобразуем ходы `из moves.txt` в буквы:

```
h3 → Z
g5 → O
f7 → C
d6 → F
f5 → N
d4 → Q
e6 → G
f4 → S
h5 → P
f6 → H
d7 → A
```

Некоторые клетки повторяются (например, конь может проходить через одну точку несколько раз), но согласно логике задания, **считаем каждую уникальную позицию только один раз.**

Собираем пароль из уникальных позиций в порядке первого появления:
h3-g5-f7-d6-f5-d4-e6-f4-h5-f6-d7 → zocfnqgspha

04

Открытие второго архива.
Используем полученный пароль:

```
unzip -P zocfnqgspha test.zip
```

Внутри находим файл `test.wav` - аудиофайл без каких-либо дополнительных подсказок.

05

Анализ WAV-файла в Audacity.
Откроем WAV-файл в программе Audacity и переключимся на режим отображения спектрограммы.

Для этого:
1. Открываем файл в Audacity
2. Нажимаем на название трека слева
3. Выбираем "Спектрограмма" (Spectrogram)

В спектрограмме обнаруживается скрытый текст:

```
XOR_KEY:2A5FA77C
```

Это ключ для XOR-шифрования в шестнадцатеричном формате.

Значит, где-то в файле спрятаны зашифрованные данные.

06

Поиск скрытых данных методом LSB.

XOR-ключ намекает на то, что данные зашифрованы. Самый распространенный метод стеганографии - это **LSB (Least Significant Bit)** - встраивание данных в младшие биты.

Проверим наличие LSB-стеганографии. Для этого можно использовать различные инструменты или написать собственный скрипт.

Используем Python-скрипт для извлечения LSB из WAV-файла:

```python
#!/usr/bin/env python3
import wave
import sys

def extract_message_from_wav(input_wav):
    # Открываем WAV файл
    audio = wave.open(input_wav, 'rb')

    # Получаем параметры
    params = audio.getparams()
    n_frames = params.nframes

    print(f"Чтение WAV файла: {input_wav}")
    print(f"Количество фреймов: {n_frames}")

    # Читаем все фреймы
    frames = bytearray(audio.readframes(n_frames))
    audio.close()

    # Извлекаем длину сообщения (первые 32 бита)
    bit_index = 0
    message_length = 0

    print("Извлечение длины сообщения...")
    for i in range(32):
        bit = frames[bit_index] & 1
        message_length = (message_length << 1) | bit
        bit_index += 1

    print(f"Длина сообщения: {message_length} байт")

    # Проверяем корректность длины
    if message_length <= 0 or message_length > len(frames) // 8:
        print("Ошибка: Некорректная длина сообщения")
        return None

    # Извлекаем байты сообщения
    message_bytes = bytearray()
```

```
print("Извлечение сообщения...")
    for byte_idx in range(message_length):
        byte_value = 0
        for bit_idx in range(8):
            bit = frames[bit_index] & 1
            byte_value = (byte_value << 1) | bit
            bit_index += 1
        message_bytes.append(byte_value)

    return message_bytes

if __name__ == "__main__":
    input_file = "hidden.wav"
    message_bytes = extract_message_from_wav(input_file)

    if message_bytes:
        print("\nИзвлеченные данные (hex):")
        hex_string = ','.join(f'{b:02x}' for b in message_bytes)
        print(hex_string)
```

Запускаем скрипт:

```
python3 lsb_extract.py
```

Результат:

```
Извлеченные данные:
hex(79,36,c5,15,44,2b,c2,17,51,6e,c9,1f,1a,32,96,12,4d,00,c4,48
,46,33,f8,1a,1a,2d,f8,1f,42,6c,d4,0f,57)
```

07

Расшифровка XOR. Теперь у нас есть:

- Зашифрованные данные (hex): `79,36,c5,15,44,2b,c2,17,51,6e,c9,1f,1a,32,96,12,4d,00,c4,48,46,33,f8,1a,1a,2d,f8,1f,42,6c,d4,0f,57`
- XOR ключ (hex): `2A5FA77C`

Для расшифровки XOR применяем побитовую операцию XOR между зашифрованными данными и ключом (ключ повторяется циклически).Напишем Python-скрипт для расшифровки:

```
#!/usr/bin/env python3

def xor_decrypt(encrypted_hex, key_hex):
    # Преобразуем hex-строки в байты
    encrypted = bytes.fromhex(encrypted_hex.replace(',', ''))
    key = bytes.fromhex(key_hex)

    # Расшифровываем XOR
    decrypted = bytearray()
    key_len = len(key)

    for i, byte in enumerate(encrypted):
        decrypted.append(byte ^ key[i % key_len])

    return decrypted
```

```
# Зашифрованные данные
encrypted =
"79,36,c5,15,44,2b,c2,17,51,6e,c9,1f,1a,32,96,12,4d,00,c4,48,46
,33,f8,1a,1a,2d,f8,1f,42,6c,d4,0f,57"

# XOR ключ
key = "2A5FA77C"

# Расшифровываем
decrypted = xor_decrypt(encrypted, key)

# Выводим результат
print("Расшифрованное сообщение:")
try:
    message = decrypted.decode('utf-8')
    print(message)
except:
    print("Не удалось декодировать как UTF-8")
    print("Hex:", decrypted.hex())
```

Запускаем скрипт расшифровки:

```
python3 xor_decrypt.py
```

Результат:

```
Расшифрованное сообщение:
Sibintek{1nc0m1ng_c4ll_f0r_ch3ss}
```


08

Полный код скрипта для расшифровки:

```
#!/usr/bin/env python3
"""
Полный скрипт для извлечения и расшифровки флага из WAV-файла
"""

import wave
import sys


def extract_lsb_from_wav(input_wav):
    """Извлечение LSB данных из WAV файла"""

    audio = wave.open(input_wav, 'rb')
    params = audio.getparams()
    n_frames = params.nframes

    frames = bytearray(audio.readframes(n_frames))
    audio.close()

    # Извлекаем длину сообщения (первые 32 бита)
    bit_index = 0
    message_length = 0

    for i in range(32):
        bit = frames[bit_index] & 1
        message_length = (message_length << 1) | bit
        bit_index += 1

    # Извлекаем байты сообщения
    message_bytes = bytearray()

    for byte_idx in range(message_length):
        byte_value = 0
        for bit_idx in range(8):
            bit = frames[bit_index] & 1
            byte_value = (byte_value << 1) | bit
            bit_index += 1
        message_bytes.append(byte_value)
```

```
    return message_bytes


def xor_decrypt(encrypted_bytes, key_hex):
    """Расшифровка XOR"""

    key = bytes.fromhex(key_hex)
    decrypted = bytearray()
    key_len = len(key)

    for i, byte in enumerate(encrypted_bytes):
        decrypted.append(byte ^ key[i % key_len])

    return decrypted


def main():
    # Извлекаем зашифрованные данные из WAV
    encrypted_data = extract_lsb_from_wav("hidden.wav")

    print("Извлеченные данные (hex):")
    print(','.join(f'{b:02x}' for b in encrypted_data))

    # Расшифровываем с помощью XOR ключа
    xor_key = "2A5FA77C"
    decrypted = xor_decrypt(encrypted_data, xor_key)

    print("\n" + "=" * 60)
    print("ФЛАГ:")
    print("=" * 60)
    print(decrypted.decode('utf-8'))
    print("=" * 60)


if __name__ == "__main__":
    main()
```

09

Применение в реальной жизни.
Данное задание демонстрирует несколько важных техник стеганографии и криптографии, которые используются в реальных сценариях:

- Полиглот-файлы. Создание файла, который одновременно является корректным файлом нескольких форматов.
Применение в реальности:
- Обход систем детектирования вредоносного ПО
- Сокрытие данных при передаче через системы контроля

- Бинарное кодирование через визуальные паттерны. Использование визуальных элементов (цвета, позиции объектов) для кодирования бинарных данных.
Применение в реальности:
- QR-коды и другие 2D-штрихкоды
- Стеганография в изображениях для передачи секретных сообщений
- Водяные знаки (watermarking) для защиты авторских прав
- Сокрытие команд для вредоносного ПО в невинных изображениях.

- Квадрат Полибия и координатное шифрование. Классический метод шифрования, где буквы заменяются координатами в таблице.
Применение в реальности:
- Современные модификации применяются в квестах и геокешинге
- Основа для более сложных шифров (PlayFair, Four-Square)

- LSB-стеганография в аудиофайлах. Встраивание данных в младшие значащие биты аудиосэмплов без заметного изменения качества звука.
Применение в реальности:
- Скрытая передача конфиденциальной информации
- Водяные знаки в музыке для отслеживания распространения
- Вредоносное ПО может использовать LSB для сокрытия command-and-control данных

- XOR-шифрование. Побитовая операция XOR между открытым текстом и ключом.
Применение в реальности:
- Обфускация конфигурационных файлов вредоносного ПО
- One-Time Pad

- Спектральная стеганография. Сокрытие информации в частотном представлении аудиосигнала. *Применение в реальности:*
- SSTV (Slow-Scan Television) - передача изображений через аудио
- Подводные акустические коммуникации

Название: Climb up

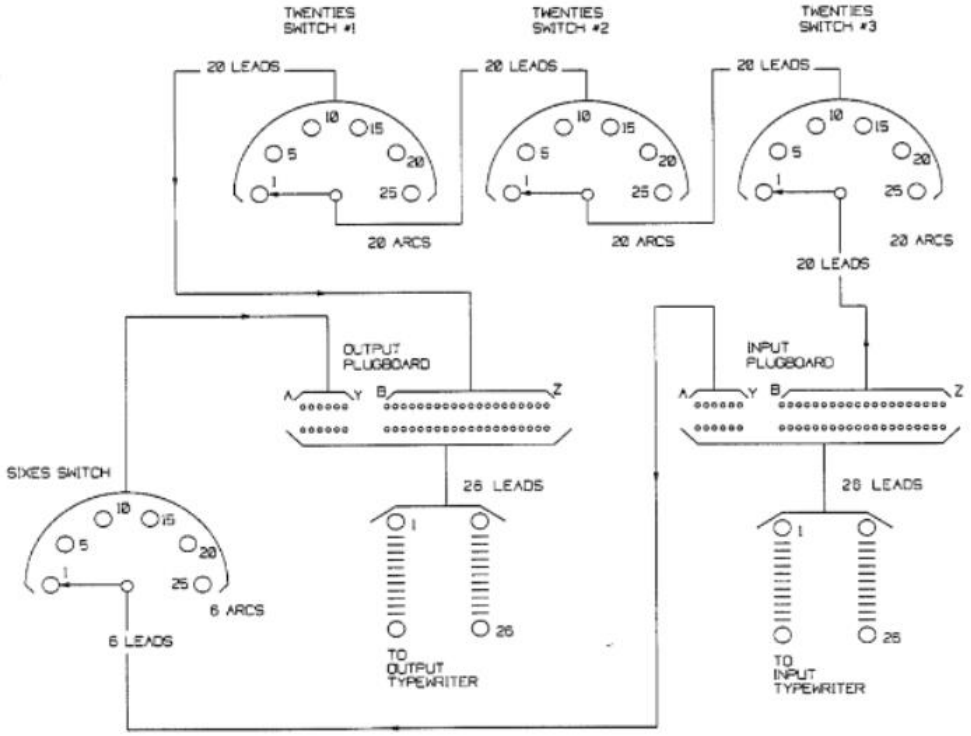
Категория: Crypto

Очки: динамическое начисление

Описание: Криптографы ДНК постоянно разрабатывают новые алгоритмы шифрования для обеспечения безопасности передаваемой на предприятии информации. Но все новое - это хорошо забытое старое, так ведь?

На столе у нашего младшего криптографа нашли записку с текстом:

"KTHAUSGKAIZFXYTMNIMXJOXOMQMQSUKPLKUSQH
AIHDEEQPFTNWNXXWJHOGHDQEXIHFQBOFEDJBQJH
IJDENKKODYDEHNRRHUWJKDTGNAZXDNLJOUKSUA
DLLSMGSMBULPJREISOMTXWSYLDCQHDMKXQIUJNK
QFPEQLPITOBYEADRSFPFKNUGQWMUGTBOXUOBML
LSPYSDTEUECAAGYKYZRONSBIJTXGNABGINVCXYSK
AJAWBNHOGEBFNSGPKZVUYJAUYPDJHTBYAQQWTC
KWCBWWXUHBJYEJRGAAPWDLWWIMIVUONBOAFNQ
ESGWIOZXYRYRTYTSMUGRJNMATAQBKETEPQDERN
AG" и вот такую схему. Кажется, он пытается разобраться с каким-то алгоритмом из прошлого, понять, насколько его легко взломать. Что же там написано? Флаг в обертке Sibintek{}



Флаг:
Sibintek{PVFCZL_MQDXRKNJBAWEGOITYUSH_1223}

01

Анализируем данную нам информацию в описание задания. Говорится о каком-то алгоритме из прошлого и о том, что его нужно взломать. Также приложена схема. Выполняем поиск по фото, находим информацию о том, что это устройство шифровальной машины Purple, которая использовалась японской армией во времена Второй мировой войны.
Помимо этого нам информации не дано.

Делаем вывод, что необходимо взломать данный шифр. Ищем информацию на тему взлома Purple в интернете. Находим статьи про алгоритм взлома Hill Climbing Attack, он основан на том, что шифр не является стойким из-за разделения ввода на блоки по 6 и 20 букв. В статьях также есть подробное описание того, из каких шагов состоит Hill Climbing Attack.

План будет таким:
1. Подготовительный этап:
- Загрузка зашифрованного текста
- Инициализация частотных таблиц (биграммы/триграммы английского)
- Определение функции оценки (scoring function)

2. Разделение на двухэтапную атаку:
- **Этап 1** - Шестерки:
• Перебор 25 позиций sixes switch
• Для каждой позиции - применяем алгоритм hill climbing над 6! = 720 перестановками sixes алфавита
• Функция оценки: биграммная частота
• Соседи генерируются swap пар букв в алфавите
• Multiple random restarts для избежания локальных максимумов
- **Этап 2** - Двадцатки:
• Используем найденные настройки шестерок
• Перебор 25×25×25×6 комбинаций (позиции + назначения fast/middle/slow)
• Для каждой комбинации - hill climbing над 20! перестановками twenties алфавита
• Функция оценки: триграммная частота
• Оптимизация: раннее прекращение неперспективных ветвей

3. Ключевые оптимизации:
- Предварительные вычисления: таблицы перестановок для всех позиций switches
- Параллелизация: независимость проверок разных конфигураций
- Эвристики остановки: прекращение при отсутствии улучшений после N итераций
- Постепенное увеличение текста: начинать с коротких отрезков для скорости

4. Критерии успеха:

- Восстановление читаемого текста
- Сходимость к максимальным scores частотных таблиц
- Стабильность решения при небольших изменениях ключа

В нашем случае алгоритм немного упростится за счет того, что нам уже дан файл с возможными алфавитами. Пишем скрипт, который реализует перебор алфавитов, подбор к каждому из них наилучшего положения роторов и их позиции.

Также учитываем хинт, в котором ставятся ограничения на позиции роторов (это позволит сделать перебор за разумное время).

```python
from purple_decryptor import PurpleDecryptor
from collections import Counter
import time
import os


class AdvancedPurpleCrackerV2:
    """Продвинутый крекер с поддержкой словаря алфавитов"""

    # Расширенный набор биграмм и триграмм
    COMMON_BIGRAMS = ['TH', 'HE', 'IN', 'ER', 'AN', 'RE', 'ON', 'AT', 'EN', 'ND',
                      'TI', 'ES', 'OR', 'TE', 'OF', 'ED', 'IS', 'IT', 'AL', 'AR']

    COMMON_TRIGRAMS = ['THE', 'AND', 'ING', 'HER', 'FOR', 'THA', 'NTH', 'INT',
                       'ERE', 'TIO', 'TER', 'EST', 'ERS', 'ATI', 'HAT', 'ATE']

    COMMON_WORDS = ['THE', 'AND', 'FOR', 'ARE', 'BUT', 'NOT', 'YOU', 'ALL', 'CAN',
                   'HER', 'WAS', 'ONE', 'OUR', 'OUT', 'DAY', 'GET', 'HAS', 'HIM',
                    'HIS', 'HOW', 'MAN', 'NEW', 'NOW', 'OLD', 'SEE', 'TIME', 'TWO',
                    'WAY', 'WHO', 'WILL', 'WITH', 'HAVE', 'THIS', 'FROM', 'THAT',
                    'THEY', 'BEEN', 'CALL', 'FIND', 'FLAG', 'CTF', 'CRYPTO']


    def __init__(self, dictionary_file=None, alphabet=None):
        self.cache = {}
        # Загружаем алфавиты из файла
        self.alphabets = self.load_dictionary(dictionary_file)
        print(f"Loaded {len(self.alphabets)} alphabets from {dictionary_file}")

    def load_dictionary(self, filepath):
        """Загружает алфавиты из файла"""
        alphabets = []

        with open(filepath, 'r') as f:
            for line in f:
                line = line.strip().upper()

                # Проверяем валидность
                if len(line) == 26 and len(set(line)) == 26 and line.isalpha():
                    alphabets.append(line)

        if not alphabets:
            raise ValueError(f"No valid alphabets found in {filepath}")

        return alphabets

    def advanced_score(self, text):
        """Улучшенная функция scoring"""
        if not text or len(text) < 10:
            return float('-inf')

        score = 0
        text = text.upper()

        # 1. Index of Coincidence (IC)
        ic = self.calculate_ic(text)
        expected_ic = 0.067
        ic_score = 1000 * (1 - abs(ic - expected_ic) / expected_ic)
        score += ic_score

        # 2. Bigram frequency
        bigram_count = 0
        for i in range(len(text) - 1):
            if text[i:i+2] in self.COMMON_BIGRAMS:
                bigram_count += 1
        score += bigram_count * 5

        # 3. Trigram frequency
        trigram_count = 0
        for i in range(len(text) - 2):
            if text[i:i+3] in self.COMMON_TRIGRAMS:
                trigram_count += 1
        score += trigram_count * 10
```

```
        # 4. Common words
        words_found = sum(1 for word in self.COMMON_WORDS if word in text)
        score += words_found * 15

        # 5. Vowel/Consonant ratio
        vowels = sum(1 for c in text if c in 'AEIOU')
        ratio = vowels / len(text) if len(text) > 0 else 0
        if 0.35 < ratio < 0.45:
            score += 100

        # 6. Repeating patterns penalty
        for i in range(len(text) - 3):
            if len(set(text[i:i+4])) == 1:
                score -= 20

        # 7. Suspicious characters
        rare_letters = sum(1 for c in text if c in 'QXZ')
        if rare_letters > len(text) * 0.05:
            score -= rare_letters * 5

        return score

    def calculate_ic(self, text):
        """Вычисляет Index of Coincidence"""
        n = len(text)
        if n <= 1:
            return 0

        freq = Counter(text)
        ic = sum(count * (count - 1) for count in freq.values())
        ic = ic / (n * (n - 1))
        return ic

    def smart_crack(self, ciphertext, max_time_seconds=300, switches_per_alphabet=4):
        """
        Умный подбор с перебором алфавитов

        Args:
            ciphertext: зашифрованный текст
            max_time_seconds: максимальное время работы (по умолчанию 300 сек = 5 мин)
            switches_per_alphabet: сколько switches тестировать для каждого алфавита
        """
        print("=" * 80)
        print("SMART CRACK MODE WITH ALPHABET DICTIONARY")
        print("=" * 80)
        print(f"\nAlphabets to test: {len(self.alphabets)}")
        print(f"Switches per alphabet: {switches_per_alphabet}")
        print(f"Total combinations: ~{len(self.alphabets) * switches_per_alphabet *
25}\n")
```

```
        start_time = time.time()

        best_score = float('-inf')
        best_result = None
        tested = 0

        # Типичные конфигурации switches для быстрой проверки
        switch_configs = [
            (1, 2, 23, '23'),   # Наиболее частая
            (1, 1, 1, '23'),
            (1, 24, 6, '23'),
            (9, 1, 24, '12'),
        ][:switches_per_alphabet]

        # Перебираем алфавиты
        for alpha_idx, alphabet in enumerate(self.alphabets, 1):
            if time.time() - start_time > max_time_seconds:
                print(f"\nTime limit reached after testing {tested} combinations")
                break

            print(f"\n[{alpha_idx}/{len(self.alphabets)}] Testing alphabet:
{alphabet[:15]}...")

            alphabet_best_score = float('-inf')
            alphabet_best = None

            # Для каждого алфавита пробуем несколько sixes позиций
            for sixes in [1, 5, 10, 15, 20]:
                if time.time() - start_time > max_time_seconds:
                    break

                # Пробуем типичные конфигурации
                for config in switch_configs:
                    switches = f"{sixes}-{config[0]},{config[1]},{config[2]}-
{config[3]}"

                    try:
                        dec = PurpleDecryptor(switches, alphabet)
                        plaintext = dec.decrypt(ciphertext)
                        score = self.advanced_score(plaintext)

                        tested += 1

                        # Лучший для этого алфавита
                        if score > alphabet_best_score:
                            alphabet_best_score = score
                            alphabet_best = {
                                'switches': switches,
                                'alphabet': alphabet,
                                'plaintext': plaintext,
                                'score': score
                            }
```

```
                        # Глобально лучший
                        if score > best_score:
                            best_score = score
                            best_result = {
                                'switches': switches,
                                'alphabet': alphabet,
                                'plaintext': plaintext,
                                'score': score
                            }
                            print(f"  ✓ NEW BEST! {switches} | Score: {score:.1f}")
                            print(f"    {plaintext[:70]}...")

                    except Exception as e:
                        pass

            # Показываем лучший результат для этого алфавита
            if alphabet_best:
                print(f"  → Best for this alphabet: {alphabet_best['switches']} |
Score: {alphabet_best_score:.1f}")

        elapsed = time.time() - start_time
        print(f"\n{'=' * 80}")
        print(f"Tested {tested} combinations in {elapsed:.1f}s")
        print(f"Average: {tested/elapsed:.1f} tests/second")

        # Hill climbing вокруг лучшего результата
        if best_result and elapsed < max_time_seconds:
            print(f"\n{'=' * 80}")
            print("HILL CLIMBING OPTIMIZATION")
            print("=" * 80)

            optimized = self.hill_climb(
                ciphertext,
                best_result['switches'],
                best_result['alphabet'],
                start_time,
                max_time_seconds
            )

            if optimized and optimized['score'] > best_result['score']:
                print(f"\n✓ Improved score: {best_result['score']:.1f} →
{optimized['score']:.1f}")
                best_result = optimized

        return best_result

    def hill_climb(self, ciphertext, base_switches, alphabet, start_time, max_time):
        """Hill climbing optimization вокруг найденного решения"""
```

```
        # Парсим switches
        parts = base_switches.split('-')
        sixes = int(parts[0])
        twenties = [int(x) for x in parts[1].split(',')]
        speed = parts[2]

        best_score = float('-inf')
        best_result = None

        print(f"Optimizing around {base_switches} with alphabet {alphabet[:15]}...")

        # Пробуем варианты вокруг
        for s_offset in range(-3, 4):
            if time.time() - start_time > max_time:
                break

            new_sixes = sixes + s_offset
            if not (1 <= new_sixes <= 25):
                continue

            for t_offset in [(0,0,0), (1,0,0), (0,1,0), (0,0,1),
                             (-1,0,0), (0,-1,0), (0,0,-1),
                             (1,1,0), (1,0,1), (0,1,1),
                             (-1,-1,0), (-1,0,-1), (0,-1,-1)]:

                new_twenties = [
                    max(1, min(25, twenties[0] + t_offset[0])),
                    max(1, min(25, twenties[1] + t_offset[1])),
                    max(1, min(25, twenties[2] + t_offset[2]))
                ]

                switches = f"{new_sixes}-
{new_twenties[0]},{new_twenties[1]},{new_twenties[2]}-{speed}"

                try:
                    dec = PurpleDecryptor(switches, alphabet)
                    plaintext = dec.decrypt(ciphertext)
                    score = self.advanced_score(plaintext)

                    if score > best_score:
                        best_score = score
                        best_result = {
                            'switches': switches,
                            'alphabet': alphabet,
                            'plaintext': plaintext,
                            'score': score
                        }
                        print(f"  ↑ {switches} | Score: {score:.1f}")

                except Exception:
```

```
                    pass

        return best_result

    def statistical_analysis(self, ciphertext):
        """Статистический анализ ciphertext"""
        print("\n" + "=" * 80)
        print("STATISTICAL ANALYSIS")
        print("=" * 80)

        # IC анализ
        ic = self.calculate_ic(ciphertext)
        print(f"\nIndex of Coincidence: {ic:.4f}")
        print(f"  Expected for English: ~0.067")
        print(f"  Expected for random: ~0.038")

        if ic > 0.06:
            print("  → Monoalphabetic or weak polyalphabetic")
        elif ic > 0.045:
            print("  → Polyalphabetic cipher (like Purple) ✓")
        else:
            print("  → Strong polyalphabetic or random")

        # Letter frequency
        freq = Counter(ciphertext)
        most_common = freq.most_common(6)
        print(f"\nMost common letters: {', '.join(f'{l}:{c}' for l,c in
most_common)}")

        # Repeating patterns
        print("\nRepeating patterns:")
        for length in [2, 3]:
            patterns = {}
            for i in range(len(ciphertext) - length + 1):
                pattern = ciphertext[i:i+length]
                if pattern.isalpha():
                    patterns[pattern] = patterns.get(pattern, 0) + 1

            top_patterns = sorted(patterns.items(), key=lambda x: x[1],
reverse=True)[:5]
            if top_patterns:
                print(f"  {length}-grams: {', '.join(f'{p}:{c}' for p,c in
top_patterns if c > 1)}")


def main():
    import sys
```

```
    ciphertext =
"KTHAUSGKAIZFXYTMNIMXJOXOMQMQSUKPLKUSQHAIHDEEQPFTNWNXXWJHOGHDQEXIHFQBOFEDJBQJHIJDENKKO
DYDEHNRRHUWJKDTGNAZXDNLJOUKSUADLLSMGSMBULPJREISOMTXWSYLDCQHDMKXQIUJNKQFPEQLPITOBYEADRS
FPFKNUGQWMUGTBOXUOBMLLSPYSDTEUECAAGYKYZRONSBIJTXGNABGINVCXYSKAJAWBNHOGEBFNSGPKZVUYJAUY
PDJHTBYAQQWTCKWCBWWXUHBJYEJRGAAPWDLWWIMIVUONBOAFNQESGWIOZXYRYRTYTSMUGRJNMATAQBKETEPQDE
RNAG"

    print("=" * 80)
    print("PURPLE CIPHER CRACKER")
    print("=" * 80)

    print("Custom dictionary file")

    filepath = input("Enter dictionary file path: ").strip()
    cracker = AdvancedPurpleCrackerV2(dictionary_file=filepath)

    print(f"\nCiphertext length: {len(ciphertext)}")

    cracker.statistical_analysis(ciphertext)

    result = cracker.smart_crack(ciphertext, max_time_seconds=300,
switches_per_alphabet=4)

    if result:
        print("\n" + "=" * 80)
        print("BEST RESULT")
        print("=" * 80)
        print(f"\nAlphabet: {result['alphabet']}")
        print(f"Switches: {result['switches']}")
        print(f"Score: {result['score']:.2f}")
        print(f"\nPlintext:\n{result['plaintext']}\n")

        with open('cracked_with_dictionary.txt', 'w') as f:
            f.write(f"Alphabet: {result['alphabet']}\n")
            f.write(f"Switches: {result['switches']}\n")
            f.write(f"Score: {result['score']:.2f}\n\n")
            f.write(f"Plaintext:\n{result['plaintext']}\n")

        print("✓ Result saved to cracked_with_dictionary.txt")
    else:
        print("\n✗ No valid result found")


if __name__ == "__main__":
    main()
```

02

Вывод:

```
========================================================================
============
BEST RESULT
========================================================================
============

Alphabet: PVFCZLMQDXRKNJBAWEGOITYUSH
Switches: 1-1,2,23-23
Score: 1992.84

Plaintext:
HEROYOUREALLYGOTAGUESSONHOWTOSOLVETHISITISREALLYSTRONGIHADTOTHINKFOR
ALONGTIMEABOUTHOWTODOTHISTASKITSHOULDHAVEBEENSIMPLEINITIALLYBUTYOUAR
EANCTFMASTERSOYOUWILLSOLVEITNEXTIWILLPUTINSOMETEXTTOSIMPLIFYTHEANALY
SISSOPAYNOATTENTIONTHEFLAGBYTHEWAYISTHEFINALALPHABETOFSIXESUNDERSCRI
PTTWENTIESUNDERSCRIPTSWITCHSTARTPOSITIONOFTWENTIESONEBYONEWITHOUTSEP
ARATION
```

03

Разделяем открытый текст на слова, находим фрагмент про флаг.

```
FLAGBYTHEWAYISTHEFINALALPHABETOFSIXESUNDERSCRIPTSANDTWENTIES
FLAG BY THE WAY IS THE FINAL ALPHABET OF SIXES UNDERSCRIPT
TWENTIES UNDERSCRIPT SWITCH START POSITION OF TWENTIES ONE BY
ONE WTHOUT SEPARATION
```

Флаг - это алфавит шестерок, нижнее подчеркивание, алфавит двадцаток, нижнее подчеркивание, стартовая позиция роторов двадцаток подряд без разделителей.

- Алфавит шестерок: PVFCZL,
- двадцаток: MQDXRKNJBAWEGOITYUSH,
- позиции роторов: 1 2 23.