Memoria CSAI Cifrado

Equipo KeyWhisperers

- Daniel Vicente Ramos (daniel.vicente.ramos@udc.es)
- Daniel Silva Iglesias (daniel.silva.iglesias@udc.es)

Uso

El algoritmo de python toma como input:

- Un texto cifrado: nombre-fichero-input
- Un diccionario: nombre-fichero-diccionario
- El hash de validación: nombre-fichero-hash

Los ficheros deben estar dentro del contenedor docker, preferentemente guardados en la carpeta resources. Hay una explicación detallada de como utilizar el script en el README del repositorio. Principalmente el script recibe 3 argumentos en orden, el texto cifrado, el diccionario y el hash.

Exemplo de comando

```
docker run -it --rm --entrypoint python key-whisperers ./KeyWhisperers.py
your-input-file your-dictionary-file your-hash-file
```

La palabra clave es devuelta por terminal.

Recursos

Partimos de los dos siguiente repositorios https://github.com/ichantzaras/polysub-cryptanalysis/ y https://github.com/pushkar-15/Vigenere-Cipher-Decrypter/

Des primero sacamos las frecuencias del inglés y usamos el fichero de python "kasiski", en concreto, la parte de obtener la longitud de la clave.

Para la primera aproximación del algoritmo de Kasiki nos habíamos basado en el psuedocodigo de la tesis de Matthew C. Berntsen titulada "AUTOMATING THE CRACKING OF SIMPLE CIPHERS". Utiliza un metodo de programación dinámica para reducir el tiempo tomado significativamente.

Del segundo repositorio sacamos el codigo para obtener varios candidatos de la longitud de clave.

Finalmente los adaptamos para que funcione con varios idiomas y diccionarios, ya que estaba hardcodeado para trabajar con el inglés.

Implementación y mejoras

En primer lugar, se comprueba si el texto o el alfabeto contiene "Ñ", si este es el caso automáticamente se utiliza el alfabeto español con sus frecuencias para realizar todos los cálculos posteriores. En caso contrario, se calcula la similitud del coseno con los diferentes idiomas mediante la frecuencia de palabras en el texto proporcionado y las frecuencias propias para cada letra en dichos idiomas.

Posteriormente, el algoritmo transforma todos los caracteres del texto en números (chars), lo que facilita trabajar con ellos. Luego, busca grupos de letras que se repiten y anota dónde aparecen, porque eso puede dar pistas sobre la longitud de la clave.

Con respecto al repositorio de partida hicimos una modificación en el análisis del idioma. Mediante el uso de la Transformada Rápida de Fourier (TTF) y su inversa, se puede calcular qué tan a menudo aparece cada letra en partes del texto cifrado y lo compara con la frecuencia de las letras de los diferentes idiomas. Esto permite inferir el idioma del mensaje y ajustar el texto cifrado a una clave probable. El algoritmo determina la mejor correlación para estos valores para cada letra y con ello genera la clave que posiblemente se utilizó para cifrar el mensaje original.

A continuación, seleccionamos un rango de tamaño de clave mínimo de 1 y máximo de 10, sobre el que iremos iterando en particiones de tamaño 10 ("PARTITION_LENGTH"), hasta que se encuentre la clave o hasta que se llegue a la longitud del texto. Para cada partición mediante el método de Kasiski para obtener el tamaño de la clave evaluando el máximo común divisor sobre las repeticiones de cada secuencia sobre el texto cifrado. Cuando obtenemos los tamaños de las claves, seleccionamos las 3 más probables con las que probaremos.

Además, evalúa las diferencias entre las posiciones de secuencias repetidas mediante el método de Kasiski, con el fin de adivinar la longitud de la clave. Posteriormente, identifica todos los números que pueden dividir estas diferencias y elige los más frecuentes como las longitudes de clave probables.

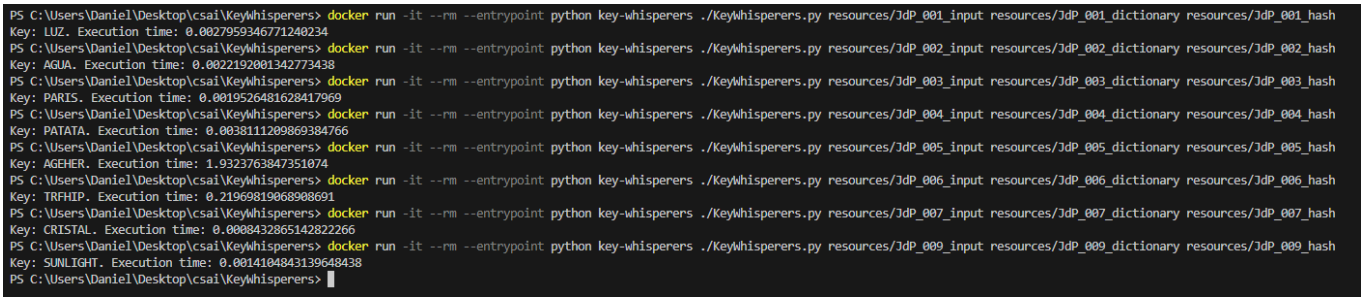
En comparación con métodos más simples, este algoritmo resulta ser sumamente eficiente y preciso. Emplea técnicas avanzadas para analizar la frecuencia de las letras, permitiendo que procese los datos de forma más rápida y exacta. Esto no solo aumenta la precisión en la identificación de la clave sino que también mejora las posibilidades de descifrar correctamente el mensaje cifrado.

Resultados

Tiempo aproximado de ejecución:

- Ejercicio 001: 2.7 ms
- Ejercicio 002: 2.2 ms
- Ejercicio 003: 1.9 ms

Tambien añadimos un par más de datos de prueba.



Posibles mejoras

- Probar solo las N longitudes más probables. Esto es algo que comenta el autor de uno de los repositorios en los que nos basamos. Hicimos pruebas y realmente no hay una mejora significativa de tiempo ya que las claves muy pequeñas. Para claves de por ejemplo 20 caracteres ya sería una mejora más significativa.