CRIPTOGRAFÍA

PRÁCTICA 1: Criptografía Clásica

AUTORES

CARLOS GARCÍA SANTA

INGENIERÍA INFORMÁTICA

ESCUELA POLITÉCNICA SUPERIOR

UNIVERSIDAD AUTÓNOMA DE MADRID

04/04/2025

Índice

1. Sustitución Monoalfabeto

a. Método afín

El script de *python afin.py* permite tanto cifrar como descifrar archivos de texto, empleando el método de cifrado afín. Este método de cifrado monoalfabético consiste en aplicar la siguiente transformación carácter a carácter:

$$y = ax + b \, mod \, m$$

donde *x* es la letra original, *y* es la letra cifrada, *a* es el coeficiente multiplicativo, *b* es el término constante, y *m* es el tamaño del alfabeto. Al estar en aritmética modular, el coeficiente *a* debe tener inverso multiplicativo dentro del anillo conmutativo Z_m para poder realizar la transformación inversa, y obtener el carácter descifrado con la expresión:

$$y = (y - b) * a^{-1} \, mod \, m$$

Para calcular el inverso primero debemos comprobar que el máximo común divisor entre el coeficiente *a* y el tamaño del alfabeto *m* es *1*, es decir que son coprimos entre sí. Para ello se utiliza el algoritmo de euclides y la biblioteca *gmpy2* que permite realizar cálculos aritméticos con alta precisión, este algoritmo se ha implementado en la función *euclidean_gcd* siguiendo las transparencias de teoría. Una vez realizada esta comprobación, se procede a aplicar el algoritmo de Euclides extendido, implementado en la función *euclidean_ext*. Este algoritmo sigue las relaciones de recurrencia establecidas en la teoría, utilizando las listas de cocientes y residuos obtenidas durante la ejecución de *euclidean_gcd*, para calcular el inverso multiplicativo de *a módulo m*, necesario para llevar a cabo el proceso de descifrado.

b. Aplicación práctica

El script de Python *afin_nt.py* implementa un método de cifrado afín no trivial que combina dos técnicas clásicas de criptografía: una permutación por sustitución seguida de una transformación afín. El método comienza aplicando una permutación del alfabeto sobre cada carácter del texto, esta permutación se define mediante una lista de enteros que representa una nueva correspondencia entre las letras del alfabeto, una vez aplicada esta sustitución, se utiliza el mismo esquema afín descrito anteriormente.

Para descifrar el texto, se realiza primero la inversión de la transformación afín, aplicando el inverso multiplicativo, y, posteriormente, se aplica la permutación inversa, es decir, se deshace la sustitución inicial para recuperar el carácter original.

El método descrito aumenta significativamente el número total de claves posibles, para demostrar esto, primero analizamos la fortaleza individual de cada uno de los criptosistemas que componen el método propuesto. El espacio de claves del cifrado afín viene definido por: $|K| = |Z_m| x |Z_m{}^*|$, done Z_m corresponde a las posibles claves de *b* (todos las letras del alfabeto) y $Z_m{}^*$ corresponde a las posibles claves de *a* (todas las letras que tienen inverso multiplicativo), por ejemplo para el inglés tenemos, $|K| = 26 * 12 = 312$. Por otro lado el cifrado de sustitución tiene la siguiente fortaleza de claves: $|K| = m!$, para el inglés: $|K| = 26!$. Entonces al realizar la composición de criptosistemas nos queda lo siguiente:

$$x = a * \pi(x) + b \bmod m \Rightarrow |K| = |Z_m| \; x \; |Z_m{}^*| \; x \; m!$$

$$|K| = 26 \; x \; 12 \; x \; 26! = 1.26 * 10^{29}$$

Sin embargo es necesario tener en cuenta que para cualquiera de las combinaciones de *a* y *b* no aparecen permutaciones nuevas, es decir, lo único que se consigue es una sustitución de una letra desplazada y el número de cifrados únicos sigue siendo $26!$, por ende, la fortaleza real del criptosistema no aumenta, y sigue siendo igual de efectivo que un cifrado de sustitución estándar. Este criptosistema se podría criptoanalizar con un ataque de texto claro conocido analizando las frecuencias de los caracteres en el texto claro y el texto cifrado para determinar las sustituciones que se están realizando, ignorando el cifrado afín, o también, si se conoce el idioma original se podría lanzar una ataque de texto cifrado conocido, lanzando hipótesis de que caracteres se convierten en otros según la frecuencia de los caracteres del texto cifrado y del idioma.

2. Sustitución Polialfabeto

a. Método de Hill

El código *hill.py* permite cifrar y descifrar archivos de texto empleando el método de cifrado de Hill, un algoritmo polialfabético basado en álgebra lineal. Este método se basa en transformar bloques de texto plano en texto cifrado mediante multiplicación matricial modular. Para ello, se divide el mensaje en bloques de tamaño *n*, luego, este bloque se multiplica por una matriz cuadrada clave *K* de dimensión $n x n$. La transformación viene definida por:

$$y = (x_1, \ldots, x_n) * K \bmod m$$

donde *x* es el bloque del texto original, *K* es la matriz clave, e *y* es el bloque cifrado. Para descifrar el mensaje, es necesario invertir la operación anterior utilizando la matriz inversa módulo *m*, denotada como K^{-1}:

$$x = (y_1, \ldots, y_n) * K^{-1} \bmod m$$

Para garantizar la existencia de esta inversa, es necesario que el determinante de *K* sea coprimo con *m*, es decir, que $gcd(det(K), m) = 1$. Esta comprobación se realiza antes de iniciar el cifrado o descifrado. En el script, esta validación se lleva a cabo utilizando la biblioteca estándar de *Python math*, y la matriz inversa modular se calcula mediante la función *inv_mod* de la librería *sympy*.

En cuanto al manejo de bloques, si el número total de caracteres alfabéticos del archivo de entrada no es múltiplo de *n*, se realiza un padding añadiendo letras aleatorias para completar el último bloque, garantizando que todos los vectores sean del mismo tamaño y puedan ser procesados, esto se mantiene para el resto de cifrados polialfabéticos (Vigenere, Transposición, Doble Permutación).

b. Método de Vigenere

El script de *Python vigenere.py* permite cifrar y descifrar archivos de texto mediante el método de Vigenere, un sistema de cifrado de tipo polialfabético. Este método utiliza una cadena de caracteres como clave, la cual se aplica de manera cíclica sobre el texto original, realizando un

desplazamiento distinto para cada letra según su posición dentro de la clave. En el código, la clave introducida por el usuario se convierte en una lista de enteros entre 0 y 25, correspondiente a la posición alfabética de cada carácter (A=0, B=1,..., Z=25). Para cifrar, se agrupan los caracteres del mensaje en bloques del mismo tamaño que la clave, y se aplica la siguiente transformación carácter a carácter:

$$y_i = (x_1 + k_1,\ldots, x_n + k_n) \, mod \, 26$$

donde x_i es la letra i-ésima sin cifrar del bloque, k_i es el valor correspondiente de la clave en la posición i, e y_i son los caracteres cifrados. Para descifrar simplemente se invierte el desplazamiento restando la clave al carácter correspondiente, de este modo, se recupera el texto original siempre que se conozca la misma clave utilizada durante el cifrado.

c. Criptoanálisis del Vigenere

Como se especifica en el enunciado se han implementado dos métodos para estimar la longitud de la clave: el método de Kasiski y el Índice de Coincidencia (IC).

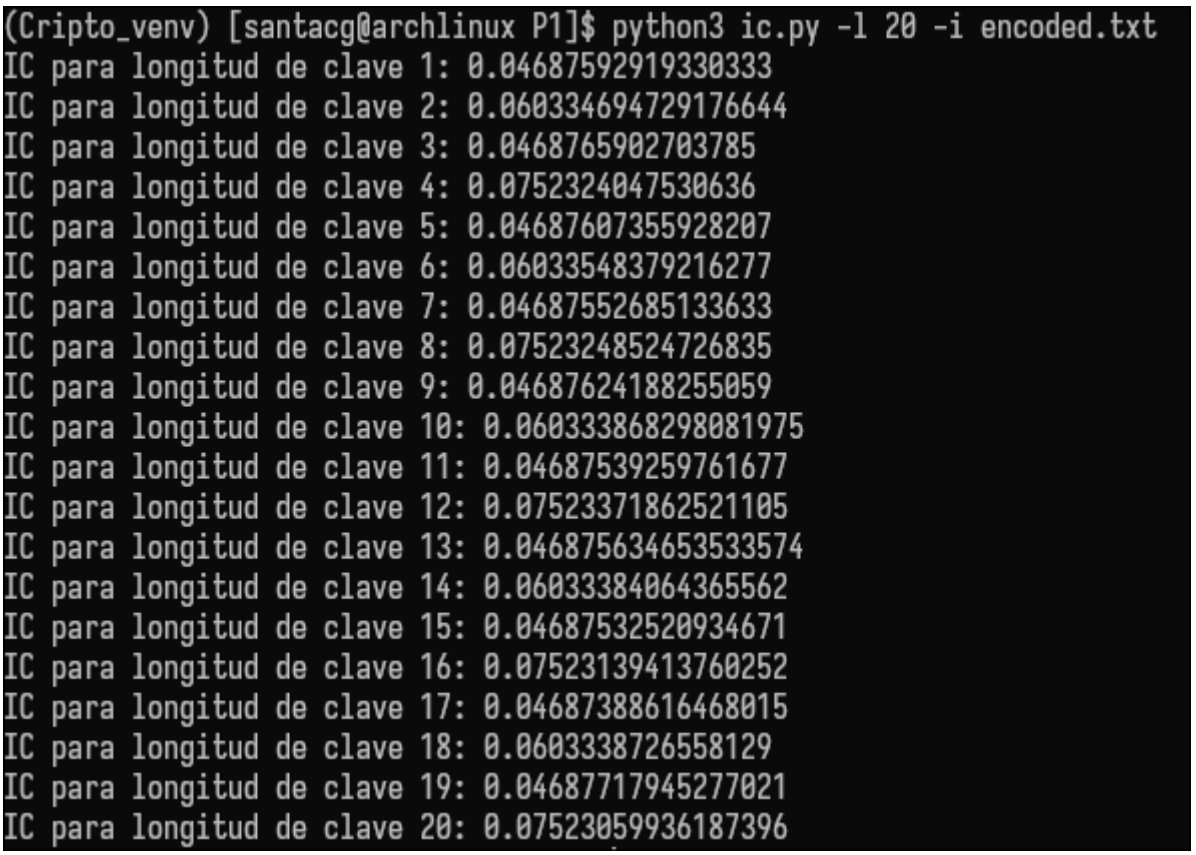
El método de kasiski, implementado en *kasiski.py*, se fundamenta en la detección de subcadenas repetidas dentro del texto cifrado. Si una misma secuencia aparece más de una vez, es probable que haya sido cifrada con la misma parte de la clave, lo cual genera repeticiones. Se selecciona un n-grama (subcadena de longitud l) y se calculan las posiciones de aparición de dicha secuencia a lo largo del texto. A continuación, se calculan las distancias entre estas posiciones, es decir, el número de caracteres que hay entre una aparición y la siguiente. La clave del método radica en que, dado que la clave se aplica de forma repetida, dichas distancias suelen ser múltiplos de la longitud de la clave. Por tanto, al obtener el MCD de todas esas distancias, se obtiene un valor que es probablemente la longitud de la clave o un divisor de ella, este es el resultado que devuelve Kasiski. A continuación se muestra un ejemplo de ejecución de Kasiski sobre *El Quijote* cifrado con Vigenere y una clave de 4 caracteres:

```
(Cripto_venv) [santacg@archlinux P1]$ python3 kasiski.py -l 1 -i encoded.txt
Longitud de clave encontrada: 2
(Cripto_venv) [santacg@archlinux P1]$ python3 kasiski.py -l 2 -i encoded.txt
Longitud de clave encontrada: 2
(Cripto_venv) [santacg@archlinux P1]$ python3 kasiski.py -l 3 -i encoded.txt
Longitud de clave encontrada: 4
(Cripto_venv) [santacg@archlinux P1]$ python3 kasiski.py -l 4 -i encoded.txt
Longitud de clave encontrada: 4
(Cripto_venv) [santacg@archlinux P1]$ python3 kasiski.py -l 5 -i encoded.txt
Longitud de clave encontrada: 4
(Cripto_venv) [santacg@archlinux P1]$ python3 kasiski.py -l 6 -i encoded.txt
Longitud de clave encontrada: 4
(Cripto_venv) [santacg@archlinux P1]$ python3 kasiski.py -l 8 -i encoded.txt
Longitud de clave encontrada: 4
(Cripto_venv) [santacg@archlinux P1]$ python3 kasiski.py -l 12 -i encoded.txt
Longitud de clave encontrada: 4
(Cripto_venv) [santacg@archlinux P1]$ python3 kasiski.py -l 15 -i encoded.txt
Longitud de clave encontrada: 4
(Cripto_venv) [santacg@archlinux P1]$ python3 kasiski.py -l 20 -i encoded.txt
Longitud de clave encontrada: 4
```

Podemos observar que Kasiski indica, cuando se eligen longitudes adecuadas de n-gramas, que la clave tiene una longitud de 4 caracteres.

El IC se basa en una propiedad estadística del idioma, que establece que la probabilidad de que dos letras elegidas al azar sean iguales es mayor en textos en lenguaje natural que en textos cifrados con claves largas. El script *ic.py* calcula el índice de coincidencia promedio para distintos valores de longitud de clave, dentro del rango 1 hasta un valor máximo definido. Para cada longitud k, se divide el texto cifrado en k subsecuencias, agrupando los caracteres que han sido cifrados con la misma letra de la clave. Se calcula el índice de coincidencia individual de cada subsecuencia y luego se obtiene el promedio. Una vez hecho esto, el valor de IC se compara con los valores esperados para idiomas no aleatorios (IC > 0.038), cuanto más alejados estén los valores de este umbral, con mayor fuerza indican una posible longitud válida de clave. A continuación se indica la fórmula empleada para el IC y un ejemplo de ejecución sobre *El Quijote* cifrado con Vigenere y una clave de 4 caracteres:

$$\frac{\sum_{i=0}^{m-1} f_i(f_i-1)}{l(l-1)}$$



Podemos observar que el IC promedio para longitudes de clave que son múltiplos de 4 es más elevado que el resto, con lo cual, elegimos cualquiera de estos valores para el cálculo de subclaves.

Una vez estimada la longitud de la clave, el script *subclaves.py* permite determinar su valor mediante un análisis de frecuencias y el índice de coincidencia. Primero, el texto cifrado se divide en subsecuencias según la longitud de la clave proporcionada, donde la primera subsecuencia contiene todas las letras cifradas con la primera letra de la clave, la segunda subsecuencia las que fueron cifradas con la segunda letra, y así sucesivamente. Para cada una de estas subsecuencias se busca encontrar el desplazamiento que se ha aplicado, es decir, la letra correspondiente de la clave. Este análisis se basa en comparar la distribución de frecuencias relativas de letras en la subsecuencia con las frecuencias típicas del idioma elegido (inglés en este caso). Para ello, se prueban los 26 posibles desplazamientos y, para cada uno, se calcula el valor:

$$M(k_i) = \sum_{j=0}^{m-1} P_j \frac{f_{j+k_i}}{\frac{l}{n}}$$

El desplazamiento óptimo se selecciona como aquel cuyo valor M(k) es más cercano al índice de coincidencia (IC ≈ 0.084, inglés en nuestro caso). La letra asociada a ese desplazamiento se toma como la letra correspondiente de la clave, repitiendo este proceso para cada subsecuencia, y el conjunto de letras resultante de esto constituye la clave de cifrado.

```
(Cripto_venv) [santacg@archlinux P1]$ python3 subclaves.py -l 4 -i encoded.txt
Subsecuencia 1:
Mejor k = 7 con M(k) = 0.0780 (diff = 0.0060)

Subsecuencia 2:
Mejor k = 14 con M(k) = 0.0778 (diff = 0.0062)

Subsecuencia 3:
Mejor k = 11 con M(k) = 0.0779 (diff = 0.0061)

Subsecuencia 4:
Mejor k = 0 con M(k) = 0.0779 (diff = 0.0061)

La clave criptoanalizada es: HOLA
```

Es importante tener en cuenta que ocurre cuando la longitud de la clave de Vigenère se aproxima a la del mensaje. En este caso, los métodos de Kasiski y el Índice de Coincidencia dejan de ser efectivos, puesto que la clave deja de repetirse con la misma periodicidad dentro del texto cifrado. Como consecuencia, dejan de observarse n-gramas repetidos a intervalos regulares, y la distribución de letras tiende a volverse uniforme, haciendo que el valor del IC converja hacia el esperado en un texto aleatorio (aproximadamente 0.038). En esta situación, el análisis de frecuencias e IC para extraer las subclaves ya no revela la estructura estadística del idioma original y no se pueden extraer las subclaves. En el caso extremo, cuando se utiliza una clave completamente aleatoria, de la misma longitud que el mensaje y usada una única vez, el esquema se convierte en un one-time pad, alcanzando el cifrado perfecto.

3. Cifrado de Flujo

El script de *Python flujo.py* permite cifrar y descifrar ficheros de texto empleando un método de cifrado de flujo basado en la generación pseudoaleatoria de claves utilizando un registro de desplazamiento con retroalimentación lineal (LFSR) tipo Fibonacci de 16 bits, descrito en (https://en.wikipedia.org/wiki/Linear-feedback_shift_register). Este método cifra el texto carácter a carácter, generando en cada paso un valor pseudoaleatorio que actúa como clave para un cifrado por desplazamiento. El algoritmo LFSR utilizado se basa en una retroalimentación lineal definida por la siguiente expresión:

$$bit\_in = (LFSR \oplus (LFSR >> 2) \oplus (LFSR >> 3) \oplus (LFSR >> 5)) \,\&\, 1$$

La función *lfsr_stream* realiza este procedimiento durante 8 iteraciones por cada carácter cifrado, generando por cada iteración un bit de salida (el bit que se saca del registro), formando así un valor entero de 8 bits (entre 0 y 255), que se utiliza como clave en el cifrado de desplazamiento de cada letra del texto.

Aunque funcional, el sistema presenta debilidades criptográficas. El LFSR tiene un tamaño de solo 16 bits, lo que significa que el número total de estados (o periodo) posibles es 65.535, además, si tenemos en cuenta que por cada carácter a cifrar se hacen 8 iteraciones, el periodo sería $65.536 / 8 = 8.192$. Esto significa que tras pasar por estos estados la secuencia o flujo cifrante se repite, con lo cual un periodo pequeño hace que la secuencia generada sea repetitiva y predecible, lo que compromete la seguridad del cifrado.

Además, al tratarse de un generador lineal, el sistema es vulnerable a ataques algebraicos. Un atacante con acceso a una pequeña parte del texto cifrado y conocimiento parcial del texto original puede deducir la secuencia clave usada, e incluso reconstruir el estado interno del LFSR utilizando el algoritmo de Berlekamp-Massey u otras técnicas de análisis de secuencias lineales. Si se reutiliza la misma secuencia para cifrar múltiples mensajes, también es posible realizar ataques diferenciales comparando las salidas.

4. Método de transposición

El script de *Python transposicion.py* implementa un método de cifrado por transposición, particularizando el cifrado de Hill mediante el uso de matrices de permutación, es decir, matrices cuadradas de dimensión nxn con un único valor 1 en cada fila y columna, y ceros en el resto. Estas matrices representan una permutación de las posiciones de un bloque de texto, sin modificar los valores de los caracteres, solo su orden. Es necesario mencionar que este tipo de matrices tienen siempre determinante igual a ±1, es decir, son siempre invertibles módulo *m* para cualquier $m > 1$, por lo que no es necesario realizar ninguna verificación sobre su invertibilidad.

5. Producto de criptosistemas de permutación

El script de *Python permutacion.py* permite cifrar y descifrar ficheros de texto utilizando un método de cifrado por doble permutación sobre bloques rectangulares de tamaño MxN. A diferencia del cifrado de Hill, en este caso no se utilizan matrices para representar la transformación, sino dos listas que definen la permutación de las filas y de las columnas, respectivamente.

El texto de entrada se divide en bloques de MxN, y a cada bloque se le aplica primero una permutación de filas seguida de una permutación de columnas. El proceso de cifrado comienza reordenando las filas de cada bloque según la lista *k1*, y posteriormente reordenando las columnas del resultado anterior según la lista *k2*. El texto cifrado se obtiene aplanando la matriz final y escribiéndola carácter a carácter.

El proceso de descifrado revierte exactamente las operaciones anteriores. A partir de las listas *k1* y *k2*, se calcula su permutación inversa, y se aplica primero la inversa de columnas y luego la inversa de filas, recuperando el orden original del texto.

Este sistema también presenta debilidades criptográficas. Al tratarse de un cifrado por transposición, los caracteres del texto no se alteran en valor, únicamente se reordenan dentro de cada bloque, como consecuencia, la distribución de frecuencias del texto original se conserva, lo que permite

aplicar ataques por análisis de frecuencia bajo el modelo de texto texto cifrado conocido. Además, si se cifran dos bloques iguales, sus versiones cifradas serán también iguales, lo cual permite detectar patrones repetidos en el texto cifrado, facilitando ataques analizando los bloques. También, si el atacante conoce o elige texto plano y su correspondiente texto cifrado, se puede deducir con facilidad la permutación aplicada.