Criptografía

ETSInf - UPV

Práctica 1

Herramientas básicas

Curso 2013-14

1. Introducción

En las primeras prácticas de la asignatura abordaremos el ataque a sistemas de cifrado monoalfabético. La principal debilidad de estos sistemas consiste en que cada uno de los símbolos del mensaje se cifran siempre del mismo modo, independientemente de la posición que ocupe en el mensaje.

Todo esto hace que una forma efectiva de atacar estos métodos sea utilizando herramientas de análisis de frecuencias. Por lo tanto, los primeros ejercicios de la primera práctica de la asignatura se dedican a implementar una colección de herramientas útiles de análisis de frecuencias. La última parte de esta práctica se dedica a la generación pseudoaleatoria de valores primos, de interés para implementaciones de sistemas de cifrado de clave pública.

Se recomienda utilizar *Mathematica* para implementar las rutinas propuestas. *Mathematica* incluye implementaciones de varias operaciones útiles en criptografía y que por lo tanto no es necesario programar. El código que pueda proporcionarse en la asignatura, como ayuda o para ilustrar algún aspecto de interés, será código *Mathematica*. Si el alumno desea utilizar otro lenguaje/entorno de programación ha de ser consciente de que, en ocasiones, no va a disponer de ciertas funcionalidades que se asume se disponen.

La utilidad de las herramientas propuestas para implementar en esta práctica está, en ocasiones, poco justificada, quedando para más adelante, bien en clase de teoría o en el laboratorio, la justificación completa.

2. Cálculo de frecuencias

La frecuencia de aparición de cada símbolo en un criptograma, y la ordenación de k-gramas en función de su frecuencia, son una herramienta útil para criptoanalizar sistemas de cifrado sencillos.

Ejercicio 1:

> Diseña un módulo *Mathematica* que reciba una lista de símbolos y obtenga una lista ordenada de los símbolos que aparecen y la frecuencia de estos en la lista de entrada.

Ejercicio 2:

Diseña un módulo *Mathematica* que reciba una lista de símbolos y obtenga una lista ordenada de los 10 bigramas más frecuentes y el número de apariciones de estos en la lista de entrada.

Ejercicio 3:

Modifica el Ejercicio 2 y diseña un módulo *Mathematica* que reciba una lista de símbolos y un valor k y obtenga una lista ordenada de los 10 k-gramas más frecuentes así como el número de apariciones de estos en la lista de entrada.

3. Cálculo del inversos módulo n

Obviamente, en todo proceso de cifrado hay que disponer de métodos que permitan tanto el cifrado como el descifrado de los mensajes. La mayoría de los métodos de cifrado incluídos en el temario de la asignatura consideran un intervalo de valores enteros (que permite codificar los símbolos del alfabeto) y dos operaciones, la suma y el producto, por lo que necesitaremos inversos para estas operaciones.

El dominio de valores que se considera se define como:

$$Z_n = \{0, 1, 2, \ldots, n-1\}$$

Teniendo en cuenta que aplicar una de estas operaciones puede dar un resultado fuera de Z_n se define la *reducción módulo n* de un número x como x mód n, de tal forma que pueda obtenerse un valor equivalente dentro del mencionado intervalo.

Si bien, para la suma, el inverso de cualquier valor en $x \in Z_n$ existe y puede obtenerse como $n - x$, para el producto, la existencia del inverso no está asegurada. Como ejemplo, puede verse que no existe el inverso para el producto de 4 en Z_{14}, ya que no hay un valor x en Z_{14} tal que su producto por 4 de 1 como resultado.

A pesar de que existe un método para el cálculo eficiente de inversos para el producto, independientemente del valor modular, hasta el momento en que se exponga este algoritmo se propone la implementación de una solución temporal.

Ejercicio 4:

Diseña un módulo *Mathematica* que reciba un par de valores x y n y calcule el inverso para el producto de x módulo n.

4. Cálculo del índice de coincidencia

En el criptoanálisis de un texto cifrado mediante un método clásico, es útil establecer una medida de dispersión de las frecuencias de los símbolos del mensaje respecto a una distribución uniforme. En esencia, esta medida compara una distribución uniforme de los símbolos y la distribución presente en un texto dado. Teniendo en cuenta un alfabeto del español con 27 símbolos:

$$MD = \sum_{i=0}^{26} \left(p_i - \frac{1}{n} \right)^2 = \sum_{i=0}^{26} \left(p_i^2 - \frac{2p_i}{n} + \frac{1}{n^2} \right) = \sum_{i=0}^{26} \left(p_i^2 \right) - 0{,}037$$

El primer término se denomina índice de coincidencia (IC), y en un texto donde los símbolos presentan una distribución própia del castellano:

$$IC = \sum_{i=0}^{26} \left(p_i^2 \right) = 0{,}072$$

Intuitivamente, el IC mide la probabilidad de que dos símbolos tomados al azar de un texto cifrado sean iguales y puede estimarse utilizando la frecuencia de los símbolos en el criptograma:

$$IC \simeq \frac{\sum_{i=0}^{26} f_i(f_i - 1)}{N(N-1)}$$

donde f_i denota el número de ocurrencias del caracter i-esimo en un criptograma de N símbolos

Ejercicio 5:

Diseña un módulo *Mathematica* que reciba una lista de símbolos y aproxime el índice de coincidencia del texto contenido en la lista.

5. Números primos

Los números primos tienen un papel importante en el diseño de sistemas criptográficos de clave pública. Por lo que disponer de herramientas para la elección de números primos *aleatorios* y de cierto tamaño es importante, sin embargo, la comprobación de primalidad de un número implica un coste computacional importante, por lo que normalmente se recurre a test que, con cierta seguridad, determinan si un número es compuesto o no.

Uno de estos test de *pseudo-primalidad* se basa en el *Pequeño Teorema de Fermat* que establece que, si un número p es primo, entonces, para cualquier $1 \leq a \leq p-1$ se cumple que:

$$a^{p-1} \equiv 1 \pmod{p}$$

el reverso del teorema no se cumple. No obstante, el Teorema de Fermat permite implementar un test de pseudo-primalidad.

Ejercicio 6: (Fermat)

Diseña un módulo *Mathematica* que implemente el test de primalidad de acuerdo al siguiente pseudocódigo:

Entrada: Un número natural $n > 1$ (potencialmente primo)
Entrada: Un número k (que determina la fiabilidad del test)
Salida: *compuesto* si n no es primo, o *posible primo* si n cumple el test.
Método
para $i = 1$ to k **do**
  Escoger a aleatorio tal que $1 < a < n$
  si $a^{n-1} \not\equiv 1 \pmod{n}$ **entonces**
    devuelve *compuesto*
  fin si
fin para
devuelve *posible primo*
FinMétodo.

El test implementado en el ejercicio anterior permite la implementación de un algoritmo para la búsqueda aleatoria de valores primos

Ejercicio 7:

Diseña un módulo *Mathematica* que implemente un proceso de búsqueda aleatoria de valores primos de acuerdo al siguiente pseudocódigo:

Entrada: El número de bits k mínimo para el número primo
Entrada: Un número t (que determina la fiabilidad del generador)
Salida: Un número probablemente primo de k bits
Método
repetir
Generar n un número aleatorio impar entre 2^k y $2^{k+1} - 1$
si $Fermat(n, t) == posible\ primo$ **entonces**
devuelve n
fin si
fin repetir
FinMétodo.

Se ha descrito que el test de Rabin falla en caso de considerar ciertos números compuestos. El test de Miller-Rabin es básicamente una modificación de este test que corrige estas deficiencias y que se utiliza para implementar un algoritmo similar al propuesto para la búsqueda aleatoria de valores primos.

Hay estudios que analizan la probabilidad de que un valor devuelto por el generador de primos de Miller-Rabin sea en realidad compuesto. Como ejemplo, citar que un valor de 100 bits devuelto por el generador de Miller-Rabin, utilizando un parámetro de confianza de $t = 6$, tiene probabilidad menor o igual de $(\frac{1}{2})^{33}$ de ser compuesto. En lo que respecta a esta práctica, hay que tener presente que los ejercicios propuestos implementan versiones simplificadas de las propuestas de Miller-Rabin, por lo que la cota anterior ha de considerarse meramente ilustrativa.