Criptografía y Cifrado de Información 2021

Lab 03

05.agosto.2021

En este laboratorio implementaremos funciones algunos genereadores psaudo-aleatorios (PRGs), y un cifrado de flujo (*stream cipher)*.

Un generador pseudo-aleatorio de **congruencias lineales** (*Linear Congruential Generator*, LCG), se fundamenta en el siguiente pseudo-código:
Algoritmo (Linear Congruential Generator, LCG).
Input: a multiplicador, b shift, N módulo, con $a, b, N \in \mathbb{Z}^+$.
Output: un stream de kt bits (k, t a elección del usuario).
`Initialize` $x_0 \pmod N$ aleatorio.
`For i = 1 to t:`
    $x_k \equiv (ax_{k-1} + b) \pmod N$.
    `output` k bits de x_k.

Un generador pseudo-aleatorio de **Wichman-Hill**, se basa en el siguiente pseudo-código:
Algoritmo (Wichman-Hill generator).
Input: a multiplicador, b shift, N módulo, con $a, b, N \in \mathbb{Z}^+$.
Output: un stream de kt bits (k, t a elección del usuario).
`Initialize` s_1, s_2, s_3 aleatorios entre 1 y 30,000.
`For i = 1 to t:`
    $s_1 = 171 * s_1 \pmod{30269}$,
    $s_2 = 172 * s_2 \pmod{30307}$,
    $s_3 = 170 * s_3 \pmod{30323}$.
    $v = s_1/30269.0 + s_2/30307.0 + s_3/30323.0 \pmod 1$
    `output` k bits de v.

1. **LCG**
   Implementar un generador pseudo-aleatorio de congruencias lineales (*Linear Congruential Generator*, LCG). Este debe recibir como argumentos el multiplicador a, el shift b y el módulo N.

   El algoritmo debe generar una cadena de bits de longitud n, indicada por el usuario. Para ello, el generador debe repetirse de forma cíclica, de manera que en cada iteración el PRG genera una secuencia de k bits, los cuales se van concatenando hasta completar los n requeridos.

   En el sitio `https://en.wikipedia.org/wiki/Linear_congruential_generator`, pueden encontrar una tabla de algunas implementaciones populares del LCG, con los parámetros indicados a, b, N, así como el número de bits k generados en cada iteración.
   En este ejercicio, se sugiere implementar por ejemplo, el método *glibc* o alguno de los métodos *POSIX*.

2. **Wichman-Hill generator**
   Implementar un generador pseudo-aleatorio de Wichman-Hill.

   El algoritmo debe generar una cadena de bits de longitud n, indicada por el usuario. Para ello, el generador debe repetirse de forma cíclica, de manera que en cada iteración el PRG genera una secuencia de k bits, los cuales se van concatenando hasta completar los n requeridos.

Típicamente este método genera un flujo dado por un número decimal con cierta cantidad de dígitos. Dentro de su implementación, este número debe ser convertido a una cadena de bits, antes de devolver los k bits requeridos.

En el sitio `https://en.wikipedia.org/wiki/Wichmann%E2%80%93Hill` pueden encontrar una implementación clásica de este método.

3. **LFSR**
   Implementar una generador pseudo-aleatorio LFSR (*Linear Feedback Shift Register*).

   En este caso, el algoritmo debe recibir los siguientes parámetros: N la longitud de su cadena de feedback, una lista con las posiciones taps a las que se hará XOR, una semilla inicial aleatoria x_0 de longitud N, y el número de iteraciones de quemado *burning phase*.

   De nuevo, el resultado debe ser una cadena de bits de longitud n, indicada por el usuario. Para ello, el generador debe repetirse de forma cíclica, de manera que en cada iteración el PRG genera una secuencia de k bits mediante el feedback, los cuales se van concatenando hasta completar los n requeridos.

   En este caso, se puede implementar como ejemplo el llamado Fibonacci LFSR, que se indica en `https://en.wikipedia.org/wiki/Linear-feedback_shift_register`.

4. **Comparación de los PRGs**
   Elegir una imagen I en escala de grises (formato 8bits sin signo), y convertir su imagen a una cadena de bits $\mathbf{x}_I$ de longitud n.

   Con los PRGs de los ejercicos anteriores, generar cadenas de bits $\mathbf{x}_P$, pseudo-aleatorias, de la misma longitud n que la de la imagen, y hacer XOR con $\mathbf{x}_I \oplus \mathbf{x}_P$. Luego, convertir esta cadena xorada de vuelta a una imagen de 8bits, de las mismas dimensiones que I, y mostrar el resultado.

   Lo que se quiere es que prueben sus tres implementaciones, usando diferentes valores para los parámetros. Prueben con parámetros N pequeños, medianos, y grandes (módulo, en el caso LCG), y (longitud de cadena feedbacken LSFR), y que muestren ejemplo de casos donde la cadena de bits XOR esconde bien el contenido de la imagen, y otras donde la cadena XOR no esconde bien el contenido.
   Mostrar en imágenes 3 ejemplos buenos, y 3 ejemplos malos.
