Presentación

Esta segunda práctica contiene dos ejercicios. En el primero implementaremos Geffe, un cifrado en flujo basado en combinaciones no lineales de LFSRs. A continuación implementaremos Trivium, un criptosistema en flujo que no está basado en LFSRs.

Objetivos

Los objetivos de esta práctica son:

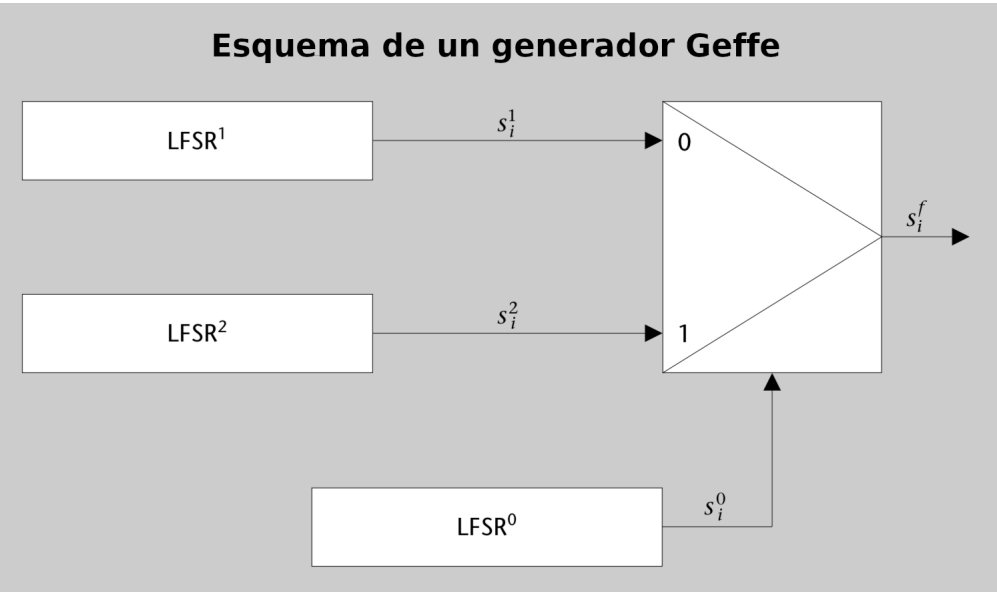
1. Familiarizarse con el uso de cripstosistemas en flujo.
2. Entender como funcionan los cifrados en flujo utilizados en la práctica.

Descripción de la Práctica a realizar

1. Implementación del cifrado en flujo de Geffe (5 puntos)

En este primer ejercicio implementaremos una función que nos permita obtener un generador pseudoaleatorio de Geffe. Posteriormente, utilizaremos este generador pseudoaleatorio para cifrar y descifrar mensajes.

El generador Geffe es un generador pseudoaleatorio formado por tres LFSRs relacionados, tal y como muestra la figura siguiente:



Observemos que hay dos LFSRs que generan dos secuencias y un tercer LFSR que determina la función de salida. De esta manera, a cada impulso de reloj $t = i$ tenemos dos valores de los LFSR^1 y LFSR^2 y otro del LFSR^0, que es el que determina cual de las dos salidas tomamos. Si la salida del LFSR^0, s_0, es un 0, se toma la salida del LFSR^1, $s_i^f = s_i^1$, mientras que si $s_o^0 = 1$, se toma la salida del LFSR^2, $s_i^f = s_i^2$.

Analíticamente el valor de la salida del generador para el impulso de reloj $t = 1$ se puede expresar de la manera siguiente:

$$s_i^f = (1 - s_i^0)s_i^1 \oplus s_i^0 s_i^2$$

donde s_i^0, s_i^1, s_i^2 , son, respectivamente, los valores que el LFSR 0, 1 y 2 generan para el impulso de reloj $t = i$.

1.1. Función que implementa un LFSR (2 puntos)

Esta función implementará el funcionamiento de un LFSR tal y como se explica en el módulo didáctico 3. La función recibirá tres parámetros: `polinomial`, `initial_state` y `num_bits`; y retornará la secuencia de salida.

- La variable `polinomial` contendrá el polinomio de conexiones del LFSR
- La variable `initial_state` contendrá el estado inicial del LFSR.
- La variable `num_bits` contendrá el número de bits de la secuencia de salida.
- La función retornará la secuencia de salida, como una cadena de caracteres de 1s y 0s.

Ejemplo:

- `polinomial`: [1,0,0,1,1] (equivale a $1 + x^3 + x^4$)
- `initial_state`: [1,0,0,0] (corresponde a $s_1 s_2 s_3 s_4$)
- `num_bits`: 20
- salida: 10001001101011110001

1.2. Función que implementa el generador pseudoaleatorio de Geffe (1,5 puntos)

Esta función implementa el generador de Geffe tal y como está definido al inicio del apartado. La función recibe como entrada cuatro parámetros: `parameters_pol_0`, `parameters_pol_1`, `parameters_pol_2` y `num_bits`; y retornará la secuencia de salida.

- Las variables `parameters_pol_x` contendrán el polinomio de conexiones y el estado inicial (en este orden) de cada un de los tres polinomios que forman el generador de Geffe.

- La variable `num_bits` contendrá el nombre de bits de la secuencia de salida.
- La función retornará la secuencia de salida, como una cadena de caracteres de 1s y 0s.

1.3. Función que cifra y descifra mensajes utilizando el generador de Geffe (1,5 puntos)

Esta función implementará un algoritmo de cifrado y descifrado en flujo usando Geffe como generador pseudoaleatorio. El procedimiento para cifrar y descifrar información se realizará mediante una operación XOR entre el mensaje y el valor de la secuencia pseudoaleatoria, tal y como se muestra en el apartado de cifrados de flujo del módulo didáctico 3.

La función tomará como variables de entrada cinco parámetros: `parameters_pol_0`, `parameters_pol_1`, `parameters_pol_2` y `message` `mode`; y retornará la secuencia de salida, que será el texto cifrado en el caso de que se esté cifrando (`mode=e`) y el texto en claro si se descifra (`mode=d`).

- Les variables `parameters_pol_x` contendrán el polinomio de conexiones y el estado inicial (en este orden) de cada uno de los tres polinomios que forman el generador de Geffe.
- La variable `message` contendrá el mensaje que se quiere cifrar o descifrar. En caso de que se quiera descifrar contendrá una cadena de 0s y 1s. En el caso de que se quiera cifrar contendrá texto en ASCII.
- La variable `mode` contendrá los valores `e` o `d` en función de si se quiere cifrar o descifrar (respectivamente).
- La función retornará el mensaje cifrado como cadena de 1s y 0s o un mensaje descifrado en ASCII.

2. Implementación del generador Trivium (5 puntos)

En esta parte de la práctica implementaremos el generador Trivium, tal y como se describe en los módulos didácticos de la asignatura. También implementaremos una función que permita cifrar y descifrar mensajes usando este generador.

2.1. Función que implementa la salida de un registro cualquiera, ya sea el A, el B o el C (1 punto)

La función recibirá como variables e entrada tres valores: `state`, `feedforward_bit` y `and_inputs`; y retornará el bit de salida del registro una vez realizado el XOR final, previo al XOR de los tres registros (ver gráfico de Trivium del módulo didáctico)

- La variable `state` contendrá el estado del registro.
- La variable `feedforward_bit` contendrá la posición de la celda del registro que se utiliza en el XOR final de la salida.

- La variable `and_inputs` contendrá las posiciones de las celdas del registro que se utilizan en el AND.
- La función retornará el bit de salida del registro y el bit de salida del AND.

2.2. Función que implementa la actualización del estado de un registro (1 punto)

Esta función recibirá como variables de entrada tres parámetros: `state`, `new_bit`, `and_prev_result` y `feedback_bit`; y retornará el nuevo estado del registro.

- La variable `state` contendrá el estado actual del registro.
- La variable `new_bit` contendrá el valor del bit procedente de la salida de otro registro.
- La variable `and_prev_result` contendrá el valor del bit procedente del AND del registro previo.
- La variable `feedback_bit` contendrá la posición de la celda del registro que se utiliza en el XOR de retroalimentación del registro.
- La función retornará el estado del registro en el siguiente impulso de reloj.

2.3. Función que implementa el generador pseudoaleatorio Trivium (2 puntos)

Esta función implementará el generador Trivium. La función tomará como variables de entrada tres parámetros: `initial_vector`, `key` y `num_bits`; y retornará la secuencia de salida.

Atención: Tened en cuenta que en el punto 4.2.1 del módulo didáctico hay una errata respecto a la implementación de Trivium. Siguiendo la publicación original, en esta práctica inicializaremos el registro A con la clave k y el registre B con el vector de inicialización VI.

- La variable `initial_vector` contendrá el vector inicial VI.
- La variable `key` contendrá la clave de cifrado k.
- La variable `num_bits` contendrá el número total de bits que se quieren generar.
- La función retornará la secuencia de salida.

2.4. Función que cifra y descifra información utilizando el generador Trivium (1 punto)

Esta función implementará un algoritmo de cifrado y descifrado en flujo utilizando el generador pseudoaleatorio Trivium. La función recibirá 4 parámetros: `message`, `key`, `iv` y `mode`; y retornará la

secuencia de salida, que será el texto cifrado en caso que se esté cifrando (`mode=e`) y el texto en claro si se está descifrando (`mode=d`).

- La variable `message` contendrá el mensaje que se quiere cifrar o descifrar. En caso de que se quiera descifrar contendrá una cadena de 0s y 1s. En el caso de que se quiera cifrar contendrá texto en ASCII.
- La variable `iv` contendrá el vector inicial VI.
- La variable `key` contendrá la clave de cifrado k.
- La variable `mode` contendrá los valores `e` o `d` en función de si se quiere cifrar o descifrar (respectivamente).
- La función retornará el mensaje cifrado como cadena de 1s y 0s o un mensaje descifrado en ASCII.

Criterios de valoración

Formato y fecha de entrega

La puntuación de cada ejercicio se encuentra detallada en el enunciado.

Por otro lado, es necesario tener en cuenta que el código que se envíe debe contener los comentarios necesarios para facilitar su seguimiento. En caso de no incluir comentarios, la corrección de la práctica se realizará únicamente de forma automática y no se proporcionará una corrección detallada. No incluir comentarios puede ser motivo de reducción de la nota.

La fecha máxima de envío es el **20/04/2020** (a las 24 horas).

Junto con el enunciado de la práctica encontrareis el esqueleto de la misma (fichero con extensión .py). Este archivo contiene las cabeceras de las funciones que hay que implementar para resolver la práctica. Este mismo archivo es el que se debe entregar una vez se codifiquen todas las funciones. No se tiene que enviar el fichero comprimido. No se aceptarán ficheros en otros foramtos que no sean .py.

Adicionalmente, también os proporcionaremos un fichero con tests unitarios para cada una de las funciones que hay que implementar. Podeis utilizar estos tests para comprobar que vuestra implementación gestiona correctamente los casos principales, así como para obtener más ejemplos concretos de lo que se espera que retornen las funciones (más allá de los que ya se proporcionan en este enunciado). Nótese, sin embargo, que los tests no son exhaustivos (no se prueban todas las entradas posibles de las funciones). Recordad que no se puede modificar ninguna parte del archivo de tests de la práctica.

La entrega de la práctica constará de un único fichero Python (extensión .py) donde se haya incluido la implementación.