Presentación

En esta práctica trabajaremos los conceptos tratados en los módulos 3 y 4 del curso, que incluyen los criptosistemas de clave simétrica y las funciones hash.

Por un lado, en la práctica trabajaremos los LFSR a través de la implementación de una extensión de uno de los criptosistemas de flujo más utilizados hoy en día, el A5/1. Por otra parte también implementaremos una función hash basada en un criptosistema de bloque.

Objetivos

Los objetivos de esta práctica son:

1. Familiarizarse con los detalles del criptosistema A5/1.
2. Implementar un criptosistema de flujo.
3. Trabajar la construcción de funciones hash.
4. Implementar una función hash basada en el criptosistema AES.

Descripción de la Pràctica a realizar

Esta práctica consta de dos partes. La primera parte se centra en los contenidos correspondientes a la criptografía de clave simétrica y la segunda parte corresponde a la temática de las funciones hash.

1. Implementación de la cifra de flujo A5/1 (5 puntos)

Tal y como se indica en los módulos didácticos de la asignatura, el algoritmo A5/1 se utiliza para cifrar las conversaciones que se realizan con la telefonía GSM. Este criptosistema es un criptosistema de flujo que utiliza una combinación no lineal de la salida de tres LFSR. Su funcionamiento se detalla en la Figura 1.

Como se puede ver, el criptosistema está formado por tres LFSR con las siguientes características:

Tamaño LFSR	Polinomio de connexiones
19	$x^{19} + x^{18} + x^{17} + x^{14} + 1$
22	$x^{22} + x^{21} + 1$
23	$x^{23} + x^{22} + x^{21} + x^8 + 1$

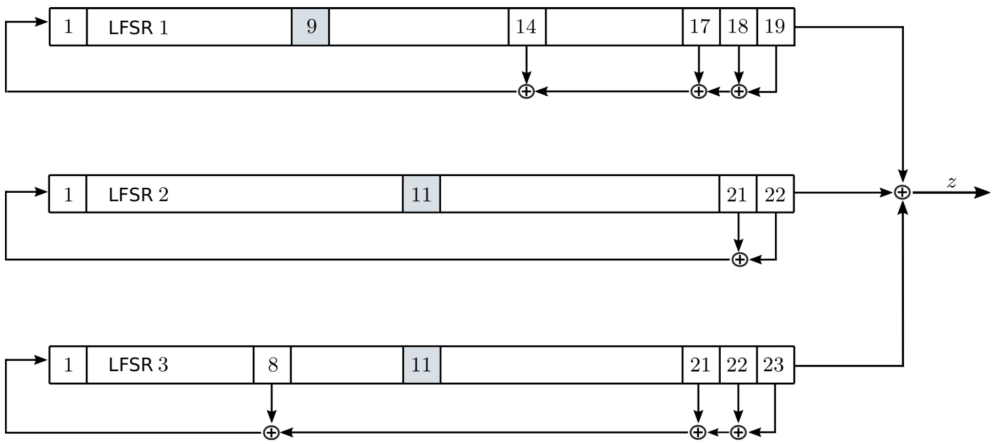


Figura 1: Esquema algoritmo A5/1.

La no linealidad del sistema viene dada porque a cada impulso de reloj no todos los LFSR avanzan. Sólo lo hacen aquellos LFSR los bits de los cuales son mayoría en las celdas denominadas clocking bit (en el caso del esquema, los clocking bits están en la celda 8 para el primer LFSR y en la celda 10 para el segundo y tercero). Por ejemplo, si en la celda 8 del primer LFSR hay un 1, y en las celdas 10 del segundo y tercer LFSR hay un 0, sólo avanzarán el segundo y el tercer LFSR, que tienen un 0. Si los tres son iguales, avanzan todos.

Para desarrollar el generador es necesario que implementéis las funciones que detallamos a continuación. Con el juego de pruebas que os proporcionamos podréis validar la corrección de vuestra implementación.

1.1. Función que implementa un LFSR. (1 punto)

La función tomará como variables de entrada tres valores: `polynomial`, `initial_state` y `output_bits`; y devolverá la secuencia de salida.

- En la variable `polynomial` se escribirá el polinomio de conexiones del LFSR;
- La variable `initial_state` contendrá el estado inicial del LFSR;
- La variable `output_bits` contendrá el número de bits de la secuencia de salida.
- La función devolverá la secuencia de salida.

Ejemplo:

- `polynomial`: `[1,1,0,0]` (equivale a x^4+x^3+1 donde el coeficiente correspondiente al término independiente no se especifica en el vector)
- `initial_state`: `[1,0,0,0]`

- `output_bits`: 20
- salida: `[1, 0, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1]`

1.2. Función que implementa una extensión del generador pseudoaleatorio A5/1. (2 puntos)

Esta función implementará una extensión del generador A5/1. Esta extensión permitirá definir un generador pseudoaleatorio como el A5/1 pero variando tanto los polinomios de conexión como los clocking bits. Es decir, el generador A5/1 será una configuración específica de la función que generaremos. La función tomará como variables de entrada cinco parámetros: `parametros_pol_0`, `parametros_pol_1`, `parametros_pol_2`, `clocking_bits` y `output_bits`; y devolverá la secuencia de salida.

- Las variables `parametros_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.
- La variable `clocking_bits` contendrá tres valores con los clocking bits de cada uno de los tres polinomios de conexiones. La función deberá verificar que los clocking bits recibidos son válidos.
- La variable `output_bits` contendrá el número total de bits de salida del generador.
- La función devolverá la secuencia de salida.

Ejemplo:

- `parametros_pol_0`: `[[1,0,1], [1,0,0]]`
- `parametros_pol_1`: `[[1,0,0,1], [1,0,0,0]]`
- `parametros_pol_2`: `[[1,0,0,1,0], [1,0,0,0,0]]`
- `clocking_bits`: `[2,3,3]`
- `output_bits`: 30
- salida: `[1, 1, 0, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1]`

1.3. Función que cifra y descifra información utilizando el generador pseudoaleatorio A5/1. (2 puntos)

Esta función implementará un algoritmo de cifrado en flujo donde el generador pseudoaleatorio será un generador A5/1. La función recibirá como variables de entrada cinco parámetros: `initial_state_0`, `initial_state_1`, `initial_state_2`, `message` y `modo`; y devolverá la secuencia de salida, que será el texto cifrado en caso de que se esté cifrando y el texto en claro si se descifra. Esta función llamará a la función desarrollada en el apartado anterior con los parámetros adecuados.

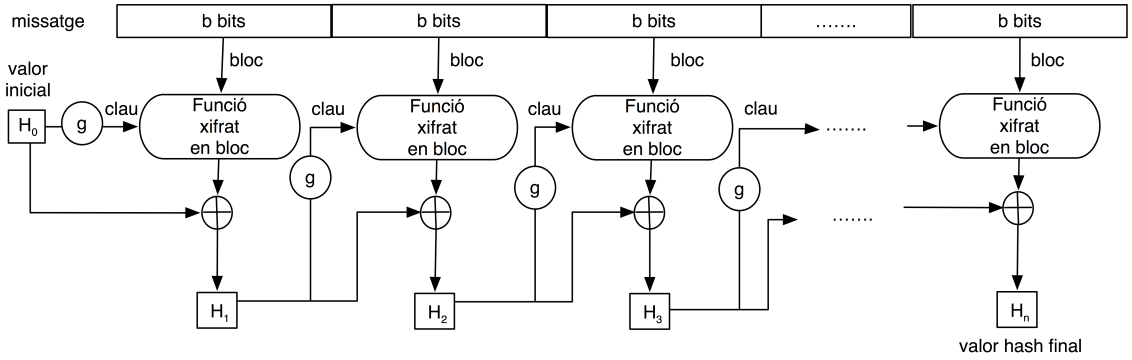
- Las variables `initial_state_X` contendrán el estado inicial de cada uno de los tres polinomios que forman el generador A5.
- La variable `message` contendrá el mensaje a tratar, el texto en claro en caso de que se quiera cifrar o el texto cifrado en caso de que se quiera descifrar. El texto en claro podrá contener un mensaje de texto de tamaño arbitrario. El texto cifrado podrá contener un mensaje de tamaño arbitrario expresado como secuencia de ceros y unos.
- La variable `mode` contendrá los valores 'e' o 'd' en función de si se quiere cifrar o descifrar (respectivamente).
- La función devolverá el mensaje obtenido.

Ejemplo:

- `initial_state_0`: `[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
- `initial_state_1`: `[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
- `initial_state_2`: `[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]`
- `message`: `'plaintext'`
- `modo`: `e`
- salida: `1101000001001100010001110100110101001100010110111000010101111100110101010`

2. Implementación de una función hash (5 puntos)

En esta parte de la práctica implementaremos una función hash basada en un criptosistema de bloque siguiendo el esquema que se muestra a continuación:



Nuestra función hash utilizará como criptosistema de bloque el AES, que cifra mensajes en bloques de 128 bits, con una clave de 256 bits. Así, nuestra función hash tratará los mensajes en bloques de $b = 128$ bits y también tendrá un tamaño de 128 bits.

Dado que el tamaño de la clave del criptosistema de bloque y el tamaño de los bloques que ciframos no son iguales, definiremos la función $g(\cdot)$ de la siguiente manera:

$$g(x) = x||x$$

donde el símbolo || denota la concatenación.

Por otra parte, el valor inicial H_0 de nuestra función hash valdrá $H_0 =$ `0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF`.

Finalmente, en caso de que el tamaño del mensaje del que queramos calcular el hash no sea un múltiplo de 128 bits, nuestra función hash hará el *padding* llenando los bits restantes con ceros.

Para desarrollar la función hash es necesario que implementéis las funciones que detallamos a continuación. Con el juego de pruebas que os proporcionamos podréis validar la corrección de vuestra implementación.

2.1. Función que implementa el criptosistema de bloque AES (1 punto)

Esta función ejecutará el algoritmo AES cifrando bloques de 128 bits con una clave de 256 bits. Para hacerlo, no implementaremos el criptosistema sino que utilizaremos la implementación que hay en la libreria pycrypto (encontraréis el import ya incluido en el esqueleto de la práctica). Dado que sólo cifraremos mensajes exactamente de tamaño 128 bits, utilizaremos AES en modo ECB. Nuestra función recibirá como variables de entrada dos parámetros: `key` y `message`; y devolverá el mensaje cifrado.

- La variable `key` contendrá la clave de cifrado de 256 bits, como una cadena de caracteres de 1 y 0s.
- La variable `message` contendrá el mensaje de 128 bits a cifrar, como una cadena de caracteres de 1 y 0s.
- La función devolverá el mensaje cifrado, como una cadena de caracteres de 1 y 0s.

2.2. Función que implementa la función $g(\cdot)$ (0.5 puntos)

Esta función implementará la función $g(\cdot)$ tal y como se ha descrito en la definición de la función hash. La función recibirá como variables de entrada un parámetro: `value` y devolverá el valor de la clave a utilizar.

- La variable `message` contendrá una cadena de 128 bits.
- La función devolverá la clave de 256 bits a utilizar.

2.3. Función que implementa el padding (0.5 puntos)

Esta función implementará el padding del mensaje. La función recibirá como variables de entrada dos parámetros: `message` y `block_len` y devolverá el mensaje con un tamaño múltiple del tamaño

de bloque.

- La variable `message` contendrá una cadena de caracteres con el mensaje.
- La variable `block_len` contendrá un entero con el tamaño de bloque.
- La función devolverá una cadena de caracteres de 1s y 0s, con la representación binaria del mensaje y tantos ceros añadidos al final de este como sean necesarios.

2.4. Función que implementa la función hash (2 puntos)

Esta función implementará la totalidad de la función hash descrita sobre un mensaje genérico de longitud arbitraria. La función recibirá como variable de entrada el parámetro `message` y devolverá el valor hash del mensaje proporcionado. Esta función llamará a las funciones desarrolladas en los apartados anteriores con los parámetros adecuados.

- La variable `message` contendrá el mensaje al que se le quiere calcular el hash.
- La función devolverá el valor hash de 128 bits correspondiente al mensaje de entrada.

2.5. Función que genera colisiones para nuestra función hash (1 punto)

La función hash que hemos implementado tiene una vulnerabilidad grave que permite a un atacante generar colisiones. Implementad una función que devuelva una colisión para nuestra función hash. La función recibirá como parámetro un prefijo, y devolverá una tupla con dos cadenas de caracteres diferentes, que empiecen por este prefijo pero que sean diferentes, y que tengan el mismo hash.

- La variable `prefijo` contendrá el prefijo que deben tener los mensajes.
- La función devolverá una tupla con dos cadenas de caracteres diferentes que empiecen por el prefijo y tengan el mismo hash.

Criterios de valoración

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

Formato y fecha de entrega

La fecha máxima de entrega de la práctica es el **05/05/2023** (a las 24 horas / hora peninsular).

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.

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.