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 de hash. Por un lado, en la práctica trabajaremos los diferentes métodos de operación de los cifrados de bloque. Por el otro, implementaremos una función de hash basada en un criptosistema de bloque.

Objetivos

Los objetivos de esta práctica son:

1. Familiaritzarse con los diferentes modos de operación de los cifrados de bloque.
2. Trabajar en la construcción de funciones de hash.
3. Implementar una función de hash basada en el criptosistema triple DES.

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 las funciones hash.

1. Implementación de diferentes modos de operación para un cifrado de bloque (5 puntos)

En esta primera parte de la práctica implementaremos los modos de operación ECB, CBC, OFB y CTR sobre el cifrado de bloque DES. Para hacerlo, no implementaremos el criptosistema DES sino que utilizaremos la implementación que hay en la librería `pycrypto` (encontraréis el *import* ya incluido en el esqueleto de la práctica). A pesar de que la función que implementa esta librería ya tiene un *flag* que permite cifrar con cada uno de estos modos de operación, en cada uno de los ejercicios de la práctica **no se pueden utilizar los *flags* de modo de operación de cifrado y sólo se podrá utilizar la función de la librería con el modelo de cifrado por defecto, que es el ECB.** El criptosistema DES es un cifrado de bloque para el que no se recomienda su uso y que está descatalogado a causa de su inseguridad, dado el tamaño de sus claves. El estándar DES tiene definido un tamaño de clave de 64 bits. Sin embargo, estos 64 bits no corresponden al tamaño efectivo de la clave, puesto que los bits correspondientes a las posiciones múltiples de 8 se usan como mecanismo de control de paridad. Por este motivo, el tamaño efectivo de las claves de DES es de 56 bits. La función DES de la librería `pycrypto`, cuando recibe los 64 bits de la clave descarta los bits de paridad.

1.1. Función que implementa el criptosistema de bloque DES en modo de operación ECB (1 punto)

Esta función ejecutará el algoritmo DES en modo de operación ECB con una clave de 56 bits. Para hacerlo, utilizaremos la implementación que hay en la librería `pycrypto`.

- La variable `key` contendrá la clave de cifrado de 56 bits, como una cadena de caracteres de 1s y 0s.
- La variable `message` contendrá el mensaje a cifrar, como una cadena de caracteres de 1s y 0s. En el caso de que el mensaje no sea múltiple del tamaño del bloque (64 bits), se realizará un *padding* rellenando los valores restantes con 0.
- La función devolverá el mensaje cifrado como una cadena de caracteres de 1s y 0s.

1.2. Función que implementa el criptosistema de bloque DES en modo de operación CBC (1 punto)

Esta función ejecutará el algoritmo DES en modo operación CBC con una calve de 56 bits. Para hacerlo usará la implementación que hay en la librería `pycrypto` **sin utilizar ningún *flag* de modo de la función**.

- La variable `IV` contendrá el vector inicial de 64 bits, como una cadena de caracteres de 1s y 0s.
- La variable `key` contendrá la clave de cifrado de 56 bits, como una cadena de caracteres de 1s y 0s.
- La variable `message` contendrá el mensaje a cifrar, como una cadena de caracteres de 1s y 0s. En el caso de que el mensaje no sea múltiple del tamaño del bloque (64 bits), se hará *padding* rellenando los valores que faltan a 0.
- La función retornará el mensaje cifrado, como una cadena de caracteres de 1s y 0s.

1.3. Función que implementa el criptosistema de bloque DES en modo de operación OFB (1,5 punto)

Esta función ejecutará el algoritmo DES en modo de operación OFB con una clave de 56 bits. Para hacerlo, utilizaremos la implementación que hay en la librería `pycrypto` **sin usar ningún *flag* de modo de la función**.

- La variable `IV` contendrá el vector inicial de 64 bits, como una cadena de caracteres de 1s y 0s.
- La variable `key` contendrá la clave de cifrado de 56 bits, como una cadena de caracteres de 1s y 0s.

- La variable `message` contendrá el mensaje a cifrar, como una cadena de caracteres de 1s y 0s. En el caso de que el mensaje no sea múltiple del tamaño del bloque (64 bits), se hará *padding* rellenando los valores restantes a 0.
- La función retornará el mensaje cifrado, como una cadena de caracteres de 1s y 0s.

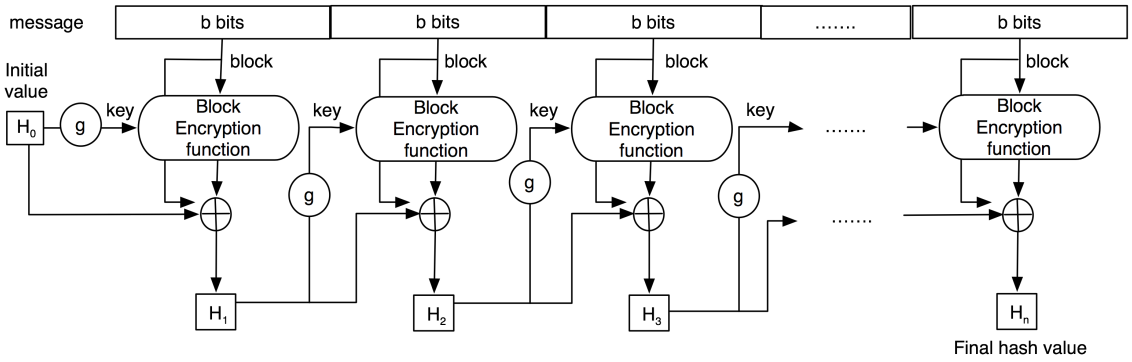
1.4. Función que implementa el criptosistema de bloque DES en modo de operación CTR (1,5 punto)

Esta función ejecutará el algoritmo DES en modo de operación CTR con una clave de 56 bits. Para hacerlo, usaremos la implementación que hay en la librería `pycrypto` **sin usar ningún flag de modo de la función**.

- La variable `nonce` contendrá un *nonce* inicial de 48 bits, como una cadena de caracteres de 1s y 0s. Los 16 bits restantes del primer valor a cifrar se usarán como contador.
- La variable `key` contendrá la clave de cifrado de 56 bits, como una cadena de caracteres de 1s y 0s.
- La variable `message` contendrá el mensaje a cifrar, como una cadena de caracteres de 1s y 0s. En el caso de que el mensaje no sea múltiple del tamaño del bloque (64 bits), se hará un *padding* rellenando los valores restantes a 0.
- La función retornará el mensaje cifrado, como una cadena de caracteres de 1s y 0s.

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 usará como criptosistema de bloque el Triple DES, que cifrará mensajes de bloques de 64 bits, utilizando DES tres veces con tres claves diferentes tal y como se indica en la

wiquipedia[1]. Esto hace que la clave efectiva de triple DES sea de 168 bits. Nuestra función hash, por lo tanto, tratará los mensajes de bloques de $b = 64$ bits y tendrá un tamaño de bloque de 64 bits.

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

$$g(x) = x_1 x_2 \cdots x_{64} 0 \cdots 0 x_{64} x_{63} \cdots x_1$$

donde $x = x_1 x_2 x_3 \cdots x_{64}$ y en el punto intermedio hay 40 ceros.

Por otro lado, el valor inicial H_0 de la nuestra función hash valdrá:

$$H_0 = \texttt{0xFFFFFFFFFFFFFFFF}$$

Finalmente, en caso de que el tamaño del mensaje del cual queramos calcular el hash no sea un múltiple de 64 bits, nuestra función hash hará el *padding* rellenando los bits restantes con unos.

Para desarrollar la función hash es necesario que implementéis las funciones que os detallamos a continuación. Con el conjunto de test que os proporcionamos podréis validar el correcto funcionamiento de vuestra implementación.

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

Esta función ejecutará el algoritmo triple DES cifrando bloques de 64 bits con una clave de 168 bits. Para hacerlo, no implementaremos el criptosistema sino que utilizaremos la implementación que hay en la llibrería `pycrypto` (encontraréis el *import* ya incluido en el esqueleto de la práctica). Dado que solo cifraremos mensajes exactamente de tamaño 64 bits, utilizaremos triple DES en modo ECB. Nuestra función recibirá como variables de entrada dos parámetros: `key` y `message`; y retornará el mensaje cifrado.

- La variable `key` contendrá la clave de cifrado de 168 bits, como una cadena de caracteres de 1s y 0s. La función triple DES de la librería indicada utiliza claves de 192 bits y descarta los bits de paridad. Por eso, hay que procesar la clave de 168 bits antes de pasarla a la función.
- La variable `message` contendrá el mensaje de 64 bits a cifrar, como una cadena de caracteres de 1s y 0s.
- La función retornará el mensaje cifrado, como una cadena de caracteres de 1s 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 variable de entrada un parámetro: `value` y retornará el valor de la clave a utilizar.

[1]Fijaros en que Triple DES no efectúa la operación de cifrado tres veces sino que cifra, el descifra y vuelve a cifrar, es decir $E_{k_1}(D_{k_2}(E_{k_3}(m)))$

- La variable `message` contendrá una cadena de 64 bits.
- La función retornará la clave de 168 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 retornará el mensaje con un tamaño múltiple del tamaño de block.

- 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 retornará una cadena de caracteres de 1s y 0s, con la representación binaria del mensaje y tantos unos añadidos al final de este como sea necesario.

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

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

- La variable `message` contendrá el mensaje para el cual se quiere calcular el hash.
- La función retornará el valor hash de 64 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 vulnerabilidat grave que permite a un atacante generar colisiones. Implementad una función que retorne una colisión para la función hash. La función recibirá como parámetro un prefijo, y retornará una tupla con dos cadenas de caracteres diferentes, que empiezan por este prefijo pero que sean diferentes, y que tengan el mismo hash.

- La variable `prefix` contendrá el prefijo que han de tener los mensajes.
- La función retornará una tupla con doss 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.

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.

Formato y fecha de entrega

La fecha máxima de envío es el **18/11/2019** (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.

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.