Universidad Nacional Autónoma de México
Facultad de Ciencias - Ciencias de la Computación
Criptografía y seguridad 2019-2

# Ejercicio 3 (que vale por dos)

En este ejercicio programarás el ataque del oráculo de padding, usando AES en modo CBC y una clave de 16 bytes.

1. Implementa el oráculo de padding, es decir, una función llamada padding_correcto que recibe un criptotexto (con el IV incluido al inicio) y devuelve un valor booleano: verdadero cuando el mensaje descifrado tiene un relleno válido y falso en caso contrario. Esta función usará AES en modo CBC con la llave global $k$ para descifrar el criptotexto, usando el IV incluido en el criptotexto. Nota que $k$ no es parte de la entrada, pues la idea es simular una caja negra que solamente responde si el descifrado tuvo un padding correcto.

2. Haz una función llamada recupera_padding que recibe una cadena $C$ de $16(n+1)$ bytes, que equivale a un bloque IV y $n$ bloques de datos, y suponemos que $M = AES_k^{-1}(C)$ contiene un padding de entre 1 y 16 bytes. La función devolverá la longitud de dicho padding. Esta función no puede usar ningún método de AES ni la llave $k$, solamente el oráculo padding_correcto.

3. Implementa una función llamada recupera_mensaje_original que recibe una cadena $C$ de $16(n+1)$ bytes, que equivale a un bloque IV y $n$ bloques de datos. La función devuelve $AES_k^{-1}(C)$ usando el modo CBC, es decir, recupera el mensaje claro que corresponde al mensaje cifrado $C$. Esta función no puede usar ningún método de AES ni la llave $k$, solamente el oráculo padding_correcto y opcionalmente la función recupera_padding.

4. Para probarlo, tu programa se deberá poder ejecutar de la siguiente forma

   ```
   $ python attack.py archivo_llave archivo_mensaje
   ```

   donde `archivo_llave` es un archivo que contiene exactamente los 16 bytes de la llave $k$, y `archivo_mensaje` contiene un mensaje cifrado cuyos primeros 16 bytes son el valor IV con el que fue cifrado. El programa guardará la salida de recupera_mensaje_original en un archivo llamado `mensaje_descifrado`.

## Notas

La clave $k$ puedes fijarla desde el inicio como una variable global, pero nota que no puede ser usada en las partes 2 y 3 del ejercicio.

Usa Python 3.x y la biblioteca PyCrypto para poder usar AES.

Los mensajes cifrados serán pequeños, por lo que puedes cargarlos en memoria completamente, pero recuerda que cifrar o descifrar archivos grandes requiere ir haciéndolo por pedazos.

Recuerda que el padding consiste en agregar entre 1 y 16 bytes al mensaje original. El valor del padding depende del tamaño requerido, el padding de un byte es la cadena `0x01`, el padding de dos bytes es la cadena `0x0202`, para tres bytes es `0x030303`, etc. Cuando se recibe un bloque y se quiere determinar si el padding es válido, nos fijamos en el último byte del bloque, que debe corresponder a un entero $1 \leq n \leq 16$, y revisamos que los últimos $n$ bytes del bloque sean iguales.

## Entrega

Adjunta tu programa y posibles observaciones en un archivo con nombre `Ejerc3_[Ape Paterno]_[Ape Materno].zip` y súbelo al Classroom. No subas archivos sueltos, solo el archivo zip.

Fecha límite: 2 de abril.