udp Escuela de Informática
y Telecomunicaciones
FACULTAD DE INGENIERÍA Y CIENCIAS

26 de mayo, 2021

Criptografía y seguridad en redes

Tarea 4: Cifrado en producción

Integrante : Joaquín Lagos Martínez

Profesor : Nicolás Boettcher
Ayudante : Macarena Velásquez

1 Características

- Python versión 3.9.5
- Librería Python: PyCryptodome versión 3.10.1
- Librería JavaScript: CryptoJS versión 4.0.0
- pip versión 21.1.2
- Algoritmo: AES
- Modo: ECB

2 Python (Servidor)

El código **Python** deberá cifrar un mensaje en texto plano con el algoritmo y modo elegido (**AES ECB**), crear un archivo **html** muy simple conteniendo el mensaje cifrado y la llave utilizada de forma que no sean visibles a primera vista. A continuación se explicará el código por partes.

Los módulos importados de **PyCryptodome** son *AES* y *pad*. La primera contiene las funciones necesarias para encriptar con **AES** y la segunda para agregar *padding* a el mensaje si es necesario.

```
1   from Crypto.Cipher import AES
2   from Crypto.Util.Padding import pad
3   import base64
```

A continuación están lo parámetros modificables. La llave (*key*) es un *string* de 16, 24 ó 32 bytes. El mensaje en texto claro (*messageToEncrypt*) es un *string*. El tamaño del bloque (*BLOCK_SIZE*) debe coincidir con el numero de bytes elegido en la contraseña y servirá como parámetro en el *padding*.

```
6   #------------------------
7   key = "01234567890123456789012345678901" # String de 16, 24 ó 32 bytes
8   messageToEncrypt = 'este es un mensaje en texto plano' # String con el mensaje en texto plano
9   BLOCK_SIZE = 32 #16, 24 ó 32 bytes, coincidiendo con el tamaño de la key
10  #------------------------
```

Para que la llave y el mensaje sean compatibles con la función de encriptado y el algoritmo en si, hay que transformarlos en bytes. Además, para que el mensaje sea compatible, la cantidad de bytes debe ser múltiplo del tamaño del bloque, por lo tanto, se le aplica un *padding pkcs7*, que viene por defecto en la función *pad*. Se le pasa como parámetros el mensaje codificado y el tamaño del bloque.

```
12  # Convirtiendo a bytes ambos parámetros
13  encodedKey = key.encode()
14  encodedMessage = messageToEncrypt.encode()
15
16  # Pad pkcs7 con tamaño de bloque como parámetro
17  paddedMessage = pad(encodedMessage, BLOCK_SIZE)
```

Aquí se inicializa el objeto AES, que recibe como parámetros la llave codificada y el modo a utilizar, en este caso, **ECB** y es la razón por la cual ser necesita *padding*. Luego se utiliza el método ***encrypt()*** con el mensaje con *pad* como argumento.

```
19  # Creando instancia que nos permitirá encriptar con AES-ECB
20  cipher = AES.new(encodedKey, AES.MODE_ECB)
21
22  # Cifrando el texto plano
23  ciphertext= cipher.encrypt(paddedMessage)
```

Para facilitar el desencriptado en **JavaScript**, se convierte el mensaje cifrado a **base64**. Y para insertarlo en un nuevo archivo **html**, se convierte a string y se le quita la 'b' y las comillas sobrantes luego de la conversión.

```
25  # Se convierte a base64 para prepararlo para desencriptar en JS
26  b64 = base64.b64encode(ciphertext)
27
28  # Preparación del resultado de la encriptación
29  # Conversión a string y eliminado de " b' " y " ' "
30  strCipher = str(b64)
31  strCipher = strCipher[2:]
32  strCipher = strCipher[:-1]
```

Finalmente, se crea el archivo 'index.html', que contiene un párrafo y dos *div*, la primera con nombre de clase 'AES' y *id* el *string* del mensaje con todas las modificaciones, y el segundo con clase 'key' y *id* la llave utilizada original.

```
34  # Creación .html con la llave y el mensaje cifrado y formateado a base64
35  f = open("index.html", "w+")
36  f.write('<p>Este sitio contiene un mensaje secreto</p>\n')
37  f.write('<div class="AES" id="'+ strCipher +'"></div>\n')
38  f.write('<div class="key" id="'+ str(key) +'"></div>')
39  f.close()
```

El archivo es creado en el directorio en el cual se ejecuta este código.

3 GitHub

GitHub permite utilizarlo como *hosting* para subir páginas web, se creó el repositorio de la tarea, se subieron los códigos de **Python**, **JavaScript** y el **html** generado. Se activó **GitHub Pages** en la configuración del repositorio y ahora se puede acceder a la 'página web', con el mensaje visible y la llave y mensaje cifrado en el código, a través del siguiente enlace.

- `https://juax16.github.io/Tarea_3_Criptografia_2021/`

4 JavaScript (Cliente)

El código en **JavaScript** se utilizará en **TamperMonkey**, el cual permite utilizar estos *scripts* sobre páginas web utilizando el ambiente del navegador. Este código debe recuperar la información de las etiquetas del código **html** y descifrar el mensaje.

En el encabezado se utiliza *@updateURL* y *@downloadURL* junto con el enlace de este código terminado en **GitHub** para que al detectarse una nueva versión en el repositorio, **TamperMonkey** te sugiera actualizarlo. Para que esto funcione, debe haber un cambio en la versión del encabezado del código. Se definió la página objetivo con *@match* y se importó la librería **CryptoJS** con *@require*.

```
7   // @updateURL    https://github.com/Juax16/Tarea_3_Criptografia_2021/blob/main/AES-ECB_decrypt.user.js
8   // @downloadURL  https://github.com/Juax16/Tarea_3_Criptografia_2021/blob/main/AES-ECB_decrypt.user.js
9   // @match        https://juax16.github.io/Tarea_3_Criptografia_2021/
10  // @require      https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.0.0/crypto-js.min.js
```

A continuación se recuperan el mensaje cifrado accediendo al **html** y buscando el elemento con clase de nombre 'AES', esto devuelve un arreglo, sabiendo que solo retorna un elemento se fija en la primera posición, y se guarda el valor del *id*. Se repite el proceso, esta vez buscando la clase 'key'.

```
20      // Se recupera el mensaje cifrado del .html
21      var cipher = document.getElementsByClassName('AES')[0].id;
22      console.log(cipher);
23
24      // Se recupera la llave de cifrado del .html
25      var key = document.getElementsByClassName('key')[0].id;
26      console.log(key);
```

La función *decrypt()* de la librería espera *WordArray* como argumentos, no *strings*. Por lo tanto son convertidos tanto la llave como el mensaje cifrado, el primero desde **utf-8** y el segundo desde **base64**. *WordArray* es propio de la librería **CryptoJS**.

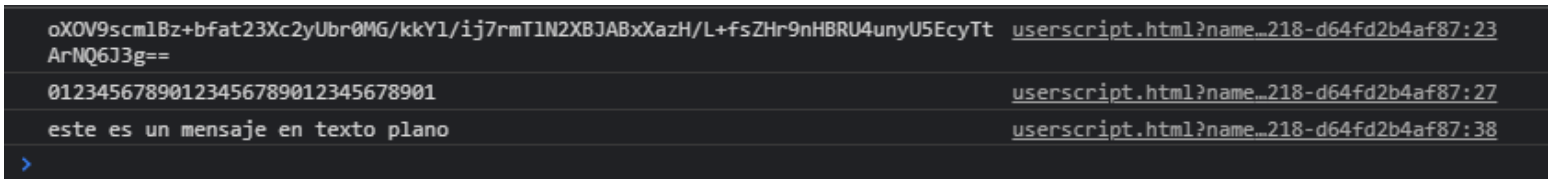
```
28      // Crea un WordArray de la llave decodificando desde utf-8
29      var wArrayKey = CryptoJS.enc.Utf8.parse(key);
30
31      // Crea un WordArray del mensaje cifrado decodificando desde base64
32      var wArrayCipher = CryptoJS.enc.Base64.parse(cipher)
```

Finalmente se usa *decrypt()*, con parámetros los *WordArray* del mensaje cifrado y la llave, además del modo a utilizar. No hay que especificar el algoritmo de *padding* ya que usa por defecto el mismo que usa la librería de **Python**. Se imprime en consola el mensaje en texto claro.

```
34      var textoPlano = CryptoJS.AES.decrypt({ciphertext: wArrayCipher}, wArrayKey,{
35          mode: CryptoJS.mode.ECB
36      })
37      console.log(textoPlano.toString(CryptoJS.enc.Utf8));
```

5 Funcionamiento:

Al ingresar a la página con el *script* en **TamperMonkey** instalado e ingresamos al inspector de elementos, podemos apreciar el mensaje recuperado en **base64**, la llave y el texto en claro en la consola.



6 GitHub

Enlace del repositorio:

- `https://github.com/Juax16/Tarea_3_Criptografia_2021/`