Tarea 2: Cifrado en producción

Sebastián Campos Vega
Criptografía y Seguridad en Redes
Escuela de Informática y Telecomunicaciones, Universidad Diego Portales
Profesor: Nicolas Boettcher
Profesor Auxiliar: Macarena Velásquez
https://github.com/ScamposV/DES-CBC/tree/DES-CBC

26 de mayo 2021

1. Introducción

Un mensaje cifrado corresponde a una transformación de cierto mensaje (texto claro) con una llave, este mensaje no es legible por quien lo vea y no posee una estructura definida.

Para cifrar se requiere un algoritmo según convenga, para este informe se utiliza el tipo de Data Encryption Standard (DES) con una extensión de Cipher Block Chaining (CBC). Los cuales para encriptar requieren ciertas características previas como, en su conjunto, una llave que llamaremos "key", un vector que nombraremos por "iv" y el mensaje a cifrar.

En consecuencia con lo anterior, se diseña un codigo en Python capaz de recibir un mensaje, un vector y una llave, donde las últimas dos variables cuentan con 8 caracteres. El programa genera un archivo HTML el cual contiene el mensaje cifrado generado por el mismo codigo. Además, se diseña un codigo en JavaScript asociado a TamperMonkey, el que es capaz de descifrar el mensaje del HTML.

2. Desarrollo

2.1. Archivo Python

2.1.1. Codigo

Este codigo se trabaja con la librería *pyDes* v2.0.1 brindando todo lo relacionado a la encriptación del mensaje. Además, se utiliza la librería *Template* para modificar los valores que serán almacenados en el archivo HTML.

En primera instancia se procede a almacenar la llave, el vector (iv) y el mensaje a cifrar, cabe destacar que la llave y el vector deben contar con 8 caracteres. Esta informacion es solicitada al usuario por el programa.

```
1  #key
2  key= input("Ingrese una llave de 8 caracteres: ")
3  while len(key) !=8:
4    key= input("Ingrese una llave de 8 caracteres, ni  m  s  ni menos: ");
5  #iv
6  iv= input("Ingrese una frase de 8 caracteres: ")
7  while len(iv) !=8:
8    iv= input("Ingrese una llave de 8 caracteres, ni  m  s  ni menos: ");
9  #mensaje a cifrar
10 text= input("Ingrese un mensaje: ")
```

Listing 1: Python - Llenado de datos

En caso de que el usuario se equivoque en la longitud tanto en la llave como en el vector, el programa le volverá a pedir las variables destacando en la limitación del largo.

Ya con esto, se procede a manipular la informacion proporcionada por el usuario para cifrar el mensaje.

```
keyb = bytes(key, 'Utf-8')
iv2= bytes(iv, 'Utf-8')
```

Listing 2: Python - Cifrando mensaje

Se transforma la llave y el vector a bytes, para que las siguientes funciones sean compatibles al momento de cifrar.

```
k = des(keyb, CBC, iv2, pad=None, padmode=PAD_PKCS5)
d = k.encrypt(text)
#hexadecimal
e=d.hex()
```

Listing 3: Python - Cifrando mensaje

Se inicializa la clase para cifrar en la variable "k", la cual recibe datos como la llave, la extensión (CBC, para este caso), pad en "None" para que no se use en todas las operaciones y por último, padmode en modo de relleno que se utiliza en todas las operaciones de cifrado. En consecuencia, se utiliza la funcion "k.encrypt" en el mensaje para cifrarlo, la que se guarda en la variable "d". Posteriormente, se guarda el mensaje encriptado en formato hexadecimal.

```
f = open('index.html','w')
html="""
<!DOCTYPE html>

<html dir="ltr" lang="en">
<head>
<meta charset="utf-8"/>
<title>Mensaje Secreto</title>
<div class= $title2 id=$vector></div>
<div class= $title id=$encrip></div>
<div class=$key id=$keygold></div>
<p>$encrip</p>
</head>
<body>
<p>Este sitio contiene un mensaje secreto</p>
<script crossorigin="anonymous" integrity="sha512-
    nOQuvD9nKirvxDdvQ9OMqe2dgapbPB7vYAMrzJihw5m+aNcf0dX53m6YxM4LgA9u8e9eg9QX+/+
    mPu8kCNpV2A==" referrerpolicy="no-referrer" src="https://cdnjs.cloudflare.com/
    ajax/libs/crypto-js/4.0.0/crypto-js.min.js"></script>
<script src="./DESCBC.js"></script>
</body>
</html>
"""
insertar=Template(html).safe_substitute(title='DESCBC',title2='iv' ,vector=iv,
    encrip=e,key='key',keygold=key)
f.write(insertar)
f.close()
```

Listing 4: Python - Creación archivo HTML

Finalmente se crea el archivo HTML desde el codigo en Python, el cual contiene como base 3 "div" los que corresponden al vector, mensaje cifrado y la llave. En la parte inferior se utiliza la funcion "Template" la cual recibe el contenido del archivo HTML y sustituye las siguientes variables por lo siguiente:

- *$title2* por iv.
- *$vector* por la variable vector que es recibida y almacenada por el programa en Python.
- *$title* por DESCBC.
- *$key* por key.

- *$keygold* por la variable key, la llave recibida y almacenada por el programa en Python.

Es efectivo el cambio luego de que se ejecuta la linea 22 con la funcion *write* y se procede a cerrar el archivo HTML.

2.1.2. Funcionamiento

```
Ingrese una llave de 8 caracteres: LLave de8
Ingrese una llave de 8 caracteres, ni más ni menos: LLavede8
Ingrese una frase de 8 caracteres: Vectorde8
Ingrese una llave de 8 caracteres, ni más ni menos: Vectord8
Ingrese un mensaje: Al fin lo logreeeee
```

Figura 1: Python - Ejecución del programa

En primera instancia, al ejecutar el programa, este solicita que el usuario escriba una llave de 8 caracteres, si el usuario no cumple con el requisito de longitud, el programa volverá a pedir el mismo valor hasta que satisfaga la condición, lo mismo pasa para la variable del vector, en este caso se pide como "frase". Finalmente, el programa pide que el usuario inserte un mensaje, el que será cifrado y expuesto en un HTML en conjunto con su llave y vector.

```
<!DOCTYPE html> == $0
<html dir="ltr" lang="en">
<head>
  <meta charset="utf-8">
  <title>Mensaje Secreto</title>
</head>
<body>
  <div class="iv" id="Vectord8"></div>
  <div class="DESCBC" id="bb736850a73066814d5b6e5adc655b460d7d96d73678af75"></div>
  <div class="key" id="LLavede8"></div>
  <p>bb736850a73066814d5b6e5adc655b460d7d96d73678af75</p>
  <p>Este sitio contiene un mensaje secreto</p>
  <script crossorigin="anonymous" integrity="sha512-nOQuvD9nKirvxDdvQ9OMqe2dgapbPB7vYAMrzJihw5m+aNcf0dX53m6YxM4LgA9u8e9eg9QX+/+mPu8kCNpV2A==" referrerpolicy="no-referrer" src="https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.0.0/crypto-js.min.js"></script>
  <script src="./DESCBC.js"></script>
</body>
</html>
```

Figura 2: HTML - Codigo generado por Python

La imagen anterior consiste en el HTML generado por el codigo en Python, donde se destaca las variables que son proporcionadas por el usuario y almacenadas por el programa. Cabe destacar que el mensaje entregado al archivo HTML es el mensaje cifrado en formato hexadecimal. Mostrándose en el HTML de la siguiente forma.

bb736850a73066814d5b6e5adc655b460d7d96d73678af75

Este sitio contiene un mensaje secreto

Figura 3: HTML - Visualización

En resumen, el programa en Python recibe una llave, un vector de 8 caracteres cada uno y un mensaje a cifrar. Se ejecutan las funciones para cifrar y se crea un archivo HTML con toda la informacion para descifrar.

Para descifrar se crea un archivo JavaScript asociado a TamperMonkey que puede almacenar los datos proporcionados por el archivo HTML y que son utilizados para el fin ya mencionado.

2.2. Archivo JavaScript

2.2.1. Codigo

Este archivo contiene toda la informacion para que el plugin de TamperMonkey funcione para descifrar el mensaje proporcionado por el HTML. Además, en cuanto al codigo en JavaScript se utiliza la librería *CryptoJS* v4.0.0 la que proporciona todas las funciones para desencriptar el mensaje.

En primera instancia se definen las sentencias que ponen en funcionamiento el plugin de TamperMonkey, luego se obtienen los valores del HTML mediante funciones get.

```
// ==UserScript==
// @name         Script cifrado
// @namespace    http://tampermonkey.net/
// @version      0.1
// @updateURL    https://github.com/ScamposV/DES-CBC/blob/asd/DESCBC.js
// @description  try to take over the world!
// @author       You
// @match        file:///C:/Users/Tat%C3%A1n/index.html
// @icon         https://www.google.com/s2/favicons?domain=google.com
// @grant        none
// @require      https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.0.0/crypto-js.
    min.js
// ==/UserScript==
//const CryptoJS = require('crypto-js')
```

Listing 5: JavaScript - Sentencias para Tampermonkey

Se interviene en @update que actualiza el script cuando cambia el archivo JavaScript (JS), @match colocando la ubicación del JS y el @require que contiene la libreria crypto-js para el uso de la extensión de TamperMonkey.

```
const keyHex = CryptoJS.enc.Utf8.parse(document.getElementsByClassName('key')[0].id
    )
const iv1 = document.getElementsByClassName('iv')[0].id;
const mensaje = document.getElementsByClassName('DESCBC')[0].id;

var iv2=CryptoJS.enc.Utf8.parse(iv1);
```

Listing 6: JavaScript - Obtención de variables del HTML

El codigo interior muestra cómo se obtienen los datos alojados en el HTML. Además, se modifica el vector y la llave con un encode Utf-8, para que las funciones de descifrado sean capaces de recibir estos datos.

```
var decoded = CryptoJS.DES.decrypt(
  {ciphertext: CryptoJS.enc.Hex.parse(document.getElementsByClassName('DESCBC')[0].
      id)},
  keyHex,{
    iv: iv2,
    mode: CryptoJS.mode.CBC,
    padding: CryptoJS.pad.NoPadding});
console.log(decoded.toString(CryptoJS.enc.Utf8))
```

Listing 7: JavaScript - Obtención de variables del HTML

Se ejecuta la funcion *CryptoJS.DES.decrypt* para descifrar el mensaje oculto. Esta funcion recibe el texto a descifrar, el vector, el modo (en este caso CBC) y el padding. Finalmente, con un console.log se procede a la visualización del texto en claro.

2.2.2. Funcionamiento

Este script se ejecuta en el mismo archivo HTML, mostrando el texto en claro cuando se inspecciona la página y se accede a la opción de consola.

Figura 4: JavaScript & TamperMonkey - Resultado

La primera linea de la imagen anterior, corresponde al console.log ejecutado por el archivo JS, mientras que la segunda linea de la imagen corresponde al console.log ejecutado por el Tamper-Monkey. Como se puede evidenciar, corresponde al mismo mensaje que se envio desde un principio en el archivo Python (ver imagen 1), salvo por unos caracteres desconocidos.

3. Conclusión

Se logra cumplir el objetivo del trabajo, cifrar y descifrar un mensaje según lo requerido por el profesor. Mas al detalle, se logra cifrar con Python y descifrar con JavaScript, mediante la utilización de un HTML.

Aparte de lo teórico, la poca información sobre esta dupla (DES-CBC) fue uno de los mayores desafíos y al comparar algunas plataformas online para cifrar, en ninguno de los sitios se lograban los mismos resultados, dejando en duda la efectividad de los algoritmos y librerías utilizadas. Sin embargo, se logró sobrepasar las adversidades y se completó el objetivo principal. La utilización de tres lenguajes distintos y la comunicacion entre ellos, ayuda bastante a la formación general como futuro profesional, aportando nuevos conocimientos en los lenguajes más utilizados en la actualidad.

4. Bibliografía

- **S.A.**, 28 de abril 2010, *"pyDes 2.0.1"*, en `https://pypi.org/project/pyDes/#description`. Recuperado en 15 de mayo 2021.
- **Whiteman**, 07 de enero 2019, *"pyDes.py"*, en `https://github.com/twhiteman/pyDes/blob/master/pyDes.py`. Recuperado en 20 de mayo 2021.