Informe Laboratorio 4

Sección 1

Jonathan A. Cuitiño Mendoza
e-mail: jonathan.cuitino@mail.udp.cl

Noviembre de 2023

Índice

1. Descripción de actividades

Para este laboratorio, deberá utilizar Tampermonkey y la librería CryptoJS (con SRI) para lograr obtener los mensajes que le está comunicando su informante. En esta ocasión, su informante fue más osado y se comunicó con usted a través de un sitio web abierto a todo el público https://cripto.tiiny.site/.

Sólo un ojo entrenado como el suyo logrará descifrar cuál es el algoritmo de cifrado utilizado y cuál es la contraseña utilizada para lograr obtener la información que está oculta.

1. Desarrolle un plugin para tampermonkey que permita obtener la llave para el descifrado de los mensajes ocultos en la página web. La llave debe ser impresa por la consola de su navegador al momento de cargar el sitio web. Utilizar la siguiente estructura:

   - La llave es: KEY

2. En el mismo plugin, se debe detectar el patrón que permite identificar la cantidad de mensajes cifrados. Debe imprimir por la consola la cantidad de mensajes cifrados. Utilizar la siguiente estructura: Los mensajes cifrados son: NUMBER

3. En el mismo plugin debe obtener cada mensaje cifrado y descifrarlo. Ambos mensajes deben ser informados por la consola (cifrado espacio descifrado) y además cada mensaje en texto plano debe ser impreso en la página web.

   El script desarrollado debe ser capaz de obtener toda la información del sitio web (llave, cantidad de mensajes, mensajes cifrados) sin ningún valor forzado. Para verificar el correcto funcionamiento de su script se utilizará un sitio web con otro texto y una cantidad distinta de mensajes cifrados. Deberá indicar la url donde se podrá descargar su script.

   Un ejemplo de lo que se debe visualizar en la consola, al ejecutar automáticamente el script, es lo siguiente:

2. Desarrollo (Parte 1)

2.1. Detecta el cifrado utilizado por el informante

Uno de los puntos mas desafiantes del presente laboratorio fue descubrir que algoritmo fue utilizado para cifrar la información. Los algoritmos de cifrado mas comunes son AES, DES, 2DES, 3DES, etc. Por lo que tras probar cada uno de ellos y sus variantes se identifico el algoritmo de cifrado correcto: el **3DES con modo de cifrado ECB**. Posteriormente, la alumna noto que también estaba esta información en la imagen adjunta a la rubrica. Sabemos que la llave para cifrar con 3DES debe ser de 16 o 24 bytes, lo que coincide con la llave encontrada. Esta información fue clave para identificar el algoritmo de cifrado utilizado por el informante.

2.2. Logra que el script solo se gatille en el sitio usado por el informante

Para lograr que el script solo sea gatillado en la pagina que nos interesa, se deben configurar correctamente los headers, en particular la linea 6, a continuacion:

```
1  // ==UserScript==
2  // @name         Descifrar mensajes cifrados en divs
3  // @namespace    https://ejemplo.com
4  // @version      1.0
5  // @description  Descifrar mensajes cifrados en divs con 3DES
6  // @match        https://cripto.tiiny.site/*
7  // @grant        unsafeWindow
8  // @require https://cdnjs.cloudflare.com/ajax/libs/crypto-js/
9  // ==/UserScript==
```

En particular, el parametros *@match* se emplea en los meta datos para especificar a que URL's se aplicara el script. Esta directiva permite limitar la ejecución del script a ciertas paginas, en este caso, *cripto.tiiny*.

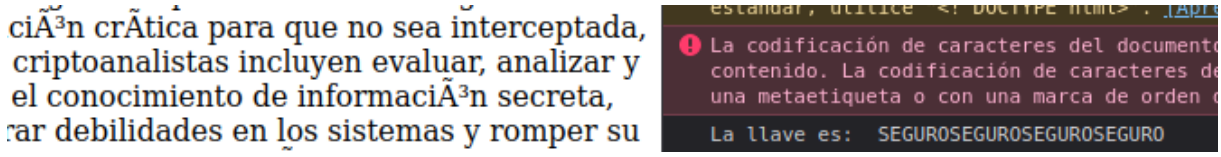
2.3. Define función que obtiene automáticamente el password del documento

Para obtener el password del documento se debe analizar el mismo en busca de cual podría ser la password oculta. Tras leer el contenido, se aprecia que hay un patrón oculto en el texto, pues al extraer solo las mayúsculas del mismo se puede ver un string coherente, y que se repite 3 veces, este string es: **SEGUROSEGUROSEGURO**. Posteriormente, y con la ayuda de chatGPT, se diseña un script en tampermonkey que permite extraer esta key e imprimirla por consola. EN particular, las lineas que realizan esta tarea son:

```
23    // Obtener todo el texto de la página
24    const textoPagina = document.body.innerText;
25
26    // Encontrar y concatenar solo las letras mayúsculas
27    const letrasMayusculas = textoPagina.match(/[A-Z]/g).join('');
28
```

2.4. Muestra la llave por consola

Una vez encontrada la clave y almacenada en la const *letrasMayusculas*, basta con hacer uso de un **console.log** para mostrar la llave por pantalla:



3. Desarrollo (Parte 2)

3.1. reconoce automáticamente la cantidad de mensajes cifrados

Tras analizar el código fuente de la pagina web auditada, se aprecia que existen mensajes en b64 cifrados en los divs del documento, particularmente en la id de cada uno de ellos. Identificada esta información, se debe entonces ampliar el script utilizado para que sea capaz de reconocer la cantidad de divs que hay en el DOM de la pagina, y de esta forma extraer sus ID's. Para lograr esto se añade la siguiente linea:

```
32    // Encontrar la cantidad de divs en el documento para luego descifrarlos
33    const divs = document.querySelectorAll('div'); // Obtener todos los divs en la página
```

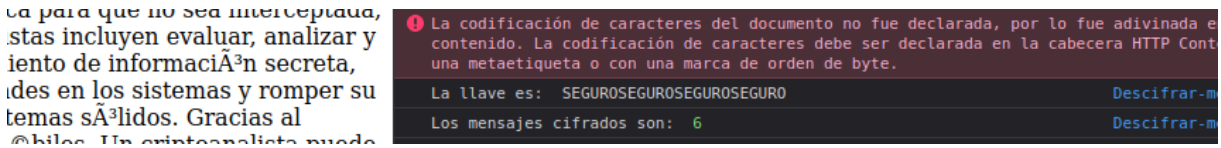
Se aprecia entonces que en la constante **divs** se almacenan los divs que posteriormente se analizaran. Para mostrar la cantidad de divs que hay en el documentos, se debe extraer el largo del string ***divs***. La evidencia en el punto consiguiente.

3.2. muestra la cantidad de mensajes por consola

Una vez almacenados los divs del DOM en la variable descrita con anterioridad, se debe obtener el largo de este arreglo para saber cuantos divs fueron identificados en la pagina. Para esto se hace uso de la funcion ***.length*** sobre el arreglo, de la siguiente manera:

```
32    // Encontrar la cantidad de divs en el documento para luego descifrarlos
33    const divs = document.querySelectorAll('div'); // Obtener todos los divs en la página
34
35    console.log("Los mensajes cifrados son: ", divs.length);
```

Al recargar la pagina, se aprecia como el script funciona correctamente, e imprime el resultado deseado por consola:



4. Desarrollo (Parte 3)

4.1. Importa la librería cryptoJS

Para descifrar los mensajes ocultos en las ID's de los divs, es necesario trabajar con la libreria Crypto-JS, la que pone a disposición del desarrollador funcionalidades capaces de cifrar y descifrar mensajes haciendo uso de distintos algoritmos. Para importar esta libreria, hacemos uso de la siguiente directiva:

```
7  // @grant        unsafeWindow
8  // @require https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.1.1/crypto-js.min.js#sha512-E8QSvWZ0eCLGk4km3hxSsNmGWbLtSCSUcewDQPQWZF6pEU8GlT8a5fF32wOl1i8ftdMhssTrF/OhyGWwonTcXA==
9  // ==/UserScript==
```

Lamentablemente, por espacio no se aprecia bien, pero a continuacion se podrá apreciar de mejor manera el comando.

```
// @require
https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.1.1/crypto-js.min.js
```

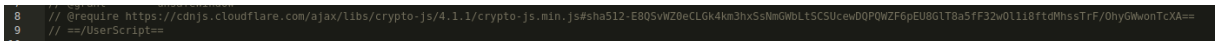
En este caso, mediante la directriz ***@require***, se puede incluir un archivo externo especificando su url; en este caso, es el script de crypto-js.

4.2. Utiliza SRI en la librería CryptoJS

Para utilizar SRI en la libreria de crypto-js es necesario ir a la pagina ***https://cdnjs.com/libraries/*** desde donde se buscara la libreria *crypto.js* y el SRI hash:

De esta forma, al presionar el ultimo icono de la derecha de la libreria en cuestión, se podrá copiar el SRI Hash. Es menester mencionar que la versión del SRI hash de crypto-js debe coincidir con la version importada en *tamperMonkey*, caso contrario no funcionara el script.

Una vez copiado el hash, este se debe concatenar en el *@require* de la libreria cripto-js, además del carácter#. De la siguiente manera:



Nuevamente, dado el largo del string, este no se aprecia bien en la imagen, pero se puede revisar en el git del laboratorio; Además, se adjunta la instrucción a continuación:

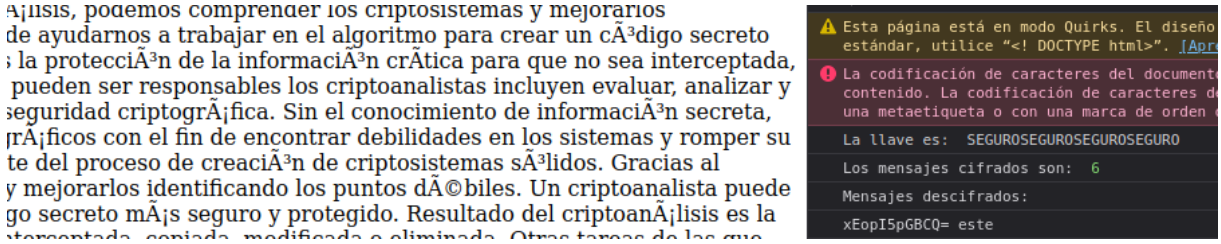
```
// @require
https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.1.1/crypto-js.min.js#sha512-
E8QSvWZ0eCLGk4km3hxSsNmGWbLtSCSUcewDQPQWZF6pEU8GlT8a5fF32wOl1i8ftdMhssTrF/Ohy
GWwonTcXA==
```

4.3. Logra descifrar uno de los mensajes

Ya habiendo añadido crypto-js y SRI al codigo de tampermonkey, se procede con el diseño de una nueva función para el script que sea capaz de descifrar el mensaje oculto en el codigo fuente de la pagina. Para esto, y con la ayuda de chatGPT se logra diseñar la función expuesta a continuación:

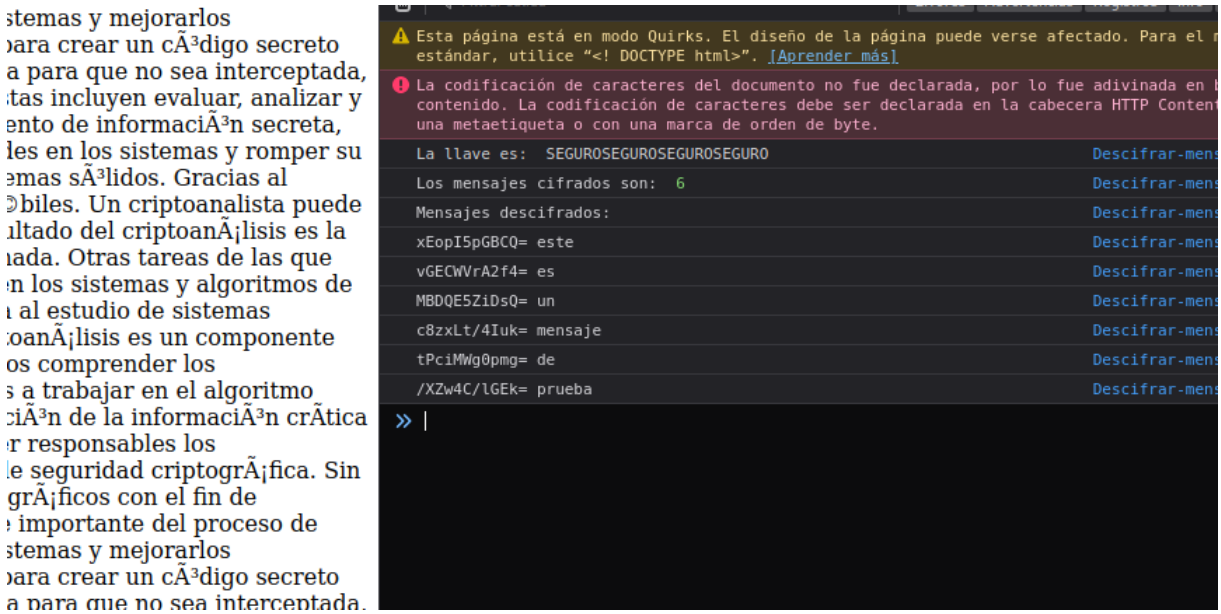
```
// Función para descifrar mensajes cifrados con 3DES ECB
function descifrarMensajeCifrado(mensajeCifrado, llave) {
    const decrypted = CryptoJS.TripleDES.decrypt({
        ciphertext: CryptoJS.enc.Base64.parse(mensajeCifrado)
    }, llave, {
        mode: CryptoJS.mode.ECB,
        padding: CryptoJS.pad.Pkcs7
    });
    return decrypted.toString(CryptoJS.enc.Utf8);
}
```

Se aprecia entonces que esta función recibe como parametros el mensaje a descifrar (en base64) y la llave. Para llegar a la verdad de que el cifrado se hizo utilizando 3DES, se debió probar variedad de algoritmos. Todos ellos, al intentan descifrar entregaban un string vació, con lo que se pasaba al siguiente algoritmo de cifrado. Hasta que finalmente, se logro descifrar correctamente con 3DES. El primer mensaje descifrado a continuacion:



4.4. Imprime todos los mensajes por consola

El código expuesto a lo largo del presente documento es capaz de obtener todos los mensajes cifrados en las id de los divs de esta pagina web, sea la cantidad que sea. Una vez el script obtiene estos mensajes cifrados, los envía a la función para descifrar, junto con la llave. Finalmente, luego de este proceso, mediante un *console.log* se imprimen por consola los mensajes cifrados y al lado el mensaje descifrado, como sigue:

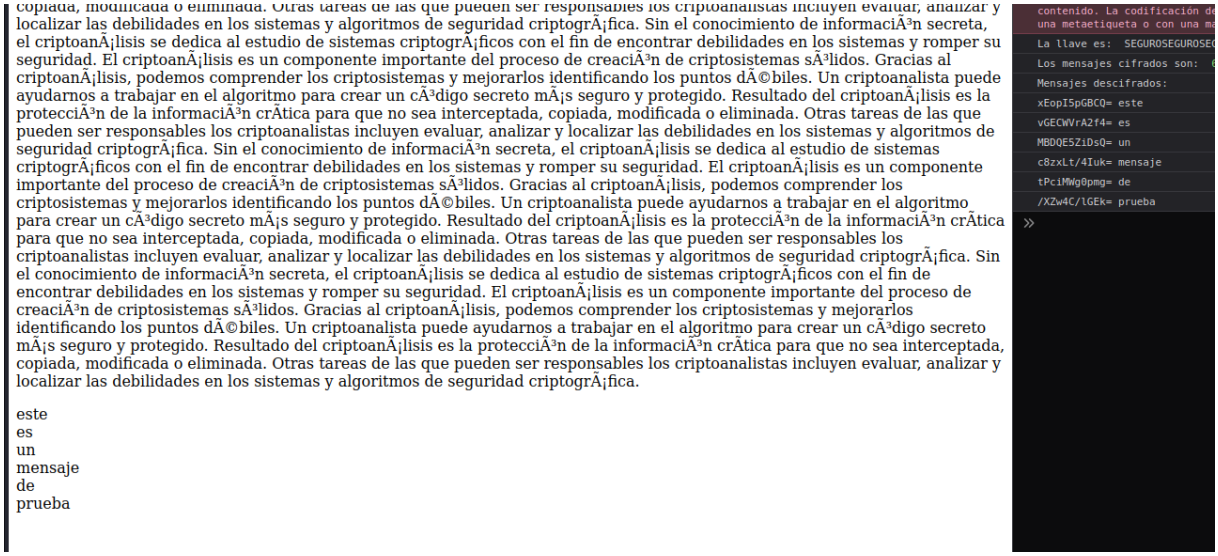


4.5. Muestra los mensajes en texto plano en el sitio web

Llegados a este punto, ya hemos encontrado cada mensaje oculto y también lo hemos descifrado, ahora se pide imprimir estos mensajes descifrados en la misma pagina web (como si fueran parte de la misma). Para esto, y haciendo uso de JS, se añade un span al DOM por cada mensaje encontrado. Para esto, se hace uso de la instrucción *document.createElement('span')*, la que creara un span por mensaje; luego, mediante la función *appendChild* se añade el texto a este span. A continuacion:

```
divs.forEach((div, index) => {
    const mensajeCifrado = div.id;
    const mensajeDescifrado = descifrarMensajeCifrado(mensajeCifrado, key);
    console.log(mensajeCifrado, mensajeDescifrado);
    const span = document.createElement('span');
    span.textContent = mensajeDescifrado;
    div.appendChild(span);
    //console.log("Mensaje", index + 1, "cifrado:", mensajeCifrado);
    //console.log("Mensaje", index + 1, "descifrado:", mensajeDescifrado);
});
```

Finalmente, los mensajes impresos en la pagina web, en texto plano:



4.6. El script logra funcionar con otro texto y otra cantidad de mensajes

A lo largo del presente documento quedo en evidencia que este script fue diseñado para funcionar con una cantidad indeterminada de divs, pero: ¿Quien nos asegura que eso realmente funciona? Para corroborar esto, se decide añadir 4 nuevos divs con ids cifradas de la misma forma (3DES con modo ECB y texto en Base 64). La forma fue añadir esto fue extendiendo el script original de la siguiente manera:

```
// Función para agregar divs con ID y clase al DOM
function agregarDivAlDOM(id, clase) {
    const newDiv = document.createElement("div");
    newDiv.id = id;
    if (clase) {
        newDiv.classList.add(clase);
    }
    document.body.appendChild(newDiv);
}
// Agregar los divs al DOM antes de continuar
agregarDivAlDOM("qm16nsFHDXM=", "M7");
agregarDivAlDOM("qm16nsFHDXM=", "M8");
agregarDivAlDOM("lAgnfs3n0ac=", "M9");
agregarDivAlDOM("ofknbazfwoY=", "M10");
```

Se aprecia del código anterior que se hace uso nuevamente de la función *appendChild*, pero esta vez añadiendo divs en vez de span.

Ahora bien, ya vimos como añadir los divs con sus respectivas ID's y las clases, pero ahora tenemos el siguiente problema: ¿Como cifrramos el mensaje con 3DES y modo ECB, para que nuestro script pueda descifrarlo? Se encontraron 3 formas de hacer esto:

- **Usando consola**: Mediante la consola de linux se puede cifrar el mensaje a usar como id del DIV a añadir. esto se hace de la siguiente manera:

```
┌──(john㉿kali)-[~]
└─$ echo -n "mejor" | openssl enc -des3 -e -a -K 53454755524f53454755524f53454755524f53454755524f -iv 0000000000000
000 -p
salt=89854D8C69550000
key=53454755524F53454755524F53454755524F53454755524F
iv =0000000000000000
lAgnfs3n0ac=
```

La instrucción a continuacion:

```
echo -n "mejor"| openssl enc -des3 -e -a -K
53454755524f53454755524f53454755524f53454755524f -iv 0000000000000000
-p
```

Se aprecia que la base se debe pasar en formato HEX y no en texto plano, por lo que mediante la misma consola se convirtio la llave **SEGUROSEGUROSEGUROSEGURO** a formato HEX, resultado: **53454755524f53454755524f53454755524f53454755524f**

- **Script en tamperMonkey:** De la misma forma en que se construyo un script en tamperMonkey para descifrar los mensajes ocultos en los DIV's, tambien se puede crear un script para ahora cifrar mensajes e imprimirlos por consola, el script expuesto a continuacion hace esa tarea:

```
// @grant        unsafeWindow
// @require https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.1.1/crypto-js.min
// ==/UserScript==

(function() {



    // Texto a cifrar
    const textoACifrar = 'equipo';

    // Clave y IV en formato Hexadecimal
    // Definir la clave en formato hexadecimal
    const keyHex = '53454755524f53454755524f53454755524f53454755524f';

    // Convertir la clave a un objeto WordArray
    const key = CryptoJS.enc.Hex.parse(keyHex);

    const iv = CryptoJS.enc.Hex.parse('0000000000000000');

    // Realizar el cifrado utilizando 3DES
    const mensajeCifrado = CryptoJS.TripleDES.encrypt(textoACifrar, key, {
        mode: CryptoJS.mode.ECB,
        padding: CryptoJS.pad.Pkcs7,
        iv: iv // Asegúrate de definir el IV si es necesario
    });

    // Mostrar el resultado del cifrado
    console.log("Mensaje cifrado:", mensajeCifrado.toString());

```

Al igual que en el caso anterior, la llave se debe pasar en formato HEX. La respuesta por consola:

```
Mensaje cifrado: ofknbazfwoY=
GET https://cripto.tiiny.site/favicon.ico
>>
```

- **Utilizando una pagina web:** También se puede cifrar un texto plano haciendo uso de alguna pagina web. Por ejemplo, la pagina `https://www.devglan.com/online-tools/`

`triple-des-encrypt-decrypt` es capas de hacer esto. A diferencia de los casos anteriores, en esta pagina la llave se debe pasar en texto plano.

4.7. Indica url al código .js implementado para su validación

Para ver los códigos utilizados, por favor revise el repositorio de gitHub que contiene los códigos y este mismo informe:

`https://github.com/iJass21/Criptografia`

Conclusiones y comentarios

El desarrollo de este laboratorio ha sido una experiencia reveladora y envolvente en el mundo de la criptografía y la seguridad en redes. A lo largo del análisis de la pagina web del informante y la implementación de scripts utilizando **Tampermonkey** y la libreria **CryptoJS** se ha logrado un entendimiento mas profundo del algoritmo 3DES, y de las técnicas para el cifrado y descifrado de información.

La identificación del algoritmo, en este caso, Triple DES (3DES) y la correcta manipulación de la llave tanto en el encriptamiento como en el desencriptamiento de la información fueron claves para la correcta realización del laboratorio.

El uso de Secure Resource Integriti ó Subresourse Integriti (SRI) es crucial para garantizar la integridad y seguridad de las herramientas utilizadas en el profeso de descifrado. Esto resalta la importancia de la seguridad en el entorno digital, especialmente al implementar funcionalidades que manipulan información sensible.

En este ejercicio, la alumna no solo ha fortalecido la comprensión de los conceptos teóricos, sino que también ha puesto de manifiesto la importancia de la práctica y la experimentación en el aprendizaje de la criptografía.