Informe Laboratorio 4

Sección 01

César Joaquín Muñoz Rivera
e-mail: cesar.munoz_r@mail.udp.cl
GitHub: Lab04 Criptografía y seguridad en redes

Junio de 2024

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

Analizando la página web proporcionada, se observa que el div principal contiene el párrafo visible al cargar la página, así como seis divs adicionales con clases Mx, donde x representa un número entero. Cada uno de estos divs posee un id codificado en Base64. Al examinar el párrafo inicial, se identificó que las letras mayúsculas ocultan un mensaje que podría servir como clave de cifrado.

A través de un análisis completo, no se encontraron vectores de inicialización (IV) que pudieran indicar el modo de operación del cifrado utilizado, descartando así los modos CBCy CFB. Además, no se encontraron contadores o nonces asociados, por lo que se descarte CTR. Sugiriendo que el cifrado opera en modo ECB y emplea una clave simétrica.

Consultando la documentación de CryptoJS [2], se identificaron DES y Triple DES como

algoritmos comunes para cifrado simétrico. Tras probar ambos algoritmos en modo ECB, se determinó que Triple DES produce resultados exitosos. Este hallazgo, respaldado por pruebas empíricas y análisis de los datos obtenidos de la página web, confirma que el cifrado utilizado es Triple DES con modo ECB.

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

Para asegurar que el script de Tampermonkey se ejecute únicamente en el sitio web proporcionado por el informante, se ha configurado el script para que se active al coincidir con la URL `https://cripto.tiiny.site/`. Esta configuración se puede observar en la línea 7 del script, como se muestra en la figura 1.

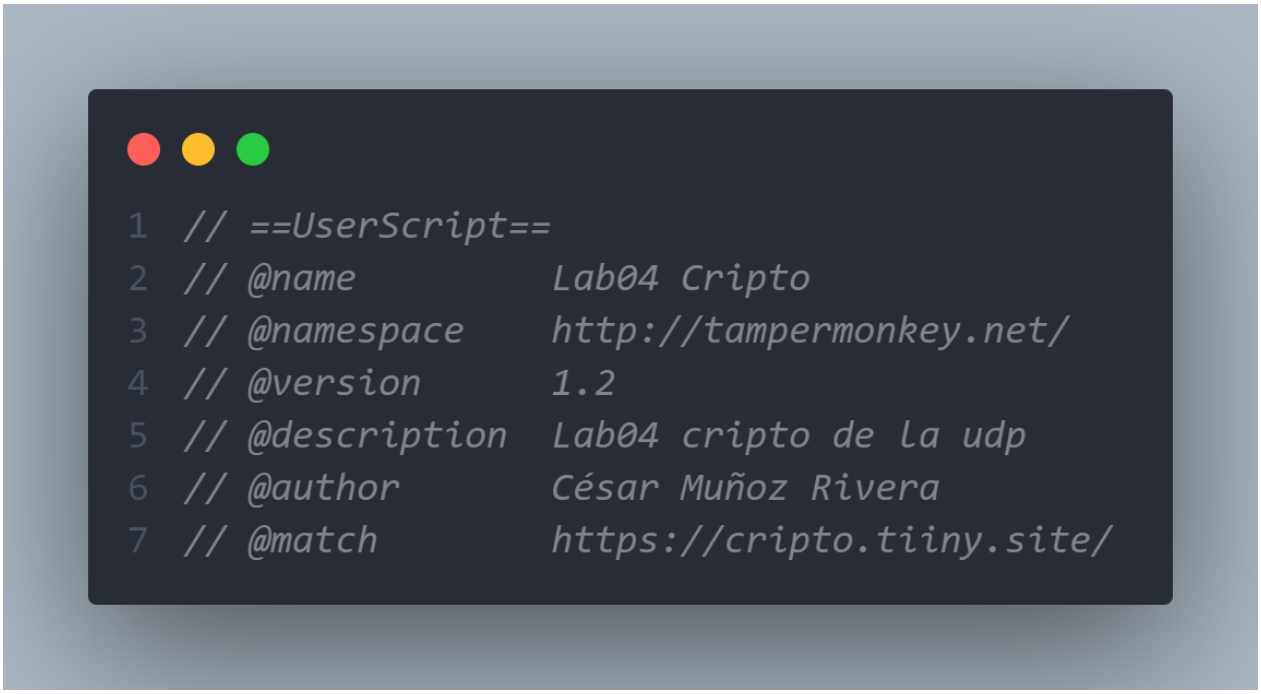

Figura 1: Configuración del script para coincidir con la URL especificada

2.3. Define función que obtiene automáticamente el password del documento

Como se explicó en el primer punto, al realizar un análisis exhaustivo de la página web, se observó que la clave (key) podría estar oculta en las letras mayúsculas del texto. Para extraer esta clave, se desarrolló una función que selecciona el elemento `<p>` y guarda todas las letras mayúsculas en una cadena de texto denominada `key`. Esta implementación se puede visualizar en la figura 2.

```
1  //Parte 1, encontrar key
2  //Seleccionar el componente <p>
3  const paragraph = document.querySelector("p");
4
5  let key = "";
6
7  if (paragraph) {
8    //Buscar todas las letras mayúsculas en el texto del párrafo
9    const uppercaseLetters = paragraph.textContent.match(/[A-Z]/g);
10
11   if (uppercaseLetters) {
12     //Crear la key a partir de las letras mayúsculas
13     key = uppercaseLetters.join("");
14     //Imprime la key
15     console.log("La llave es:", key);
16   } else {
17     console.log("No se encontraron letras mayus");
18   }
19 } else {
20   console.log("No se encontró el componente <p>.");
21 }
```

Figura 2: Función para extraer la clave del texto

2.4. Muestra la llave por consola

La función desarrollada no solo extrae la clave del documento, sino que también la imprime en la consola del navegador. Esto se puede observar en la línea 15 de la figura 2, donde la clave es registrada en la consola. La salida en la consola se muestra en la figura 3.

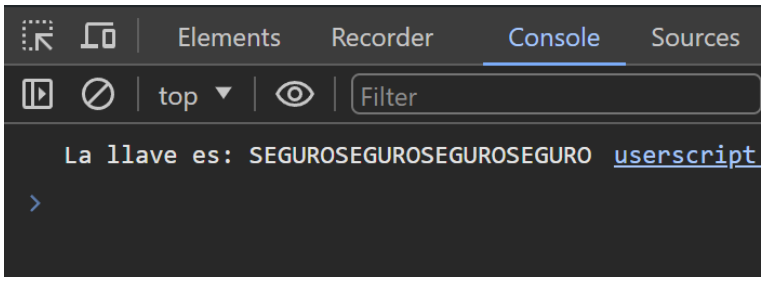

Figura 3: Impresión de la clave en la consola

3. Desarrollo (Parte 2)

3.1. Reconoce automáticamente la cantidad de mensajes cifrados

Como se mencionó anteriormente, los mensajes cifrados están contenidos en elementos div con clases del tipo Mx, donde x es un número entero. Para contar automáticamente la cantidad de estos mensajes cifrados, se implementó una parte del código que busca y cuenta los divs con clase M. Esta implementación se puede observar en la figura 4.

```
1  //Parte 2, contar cantidad de mensajes secretos (clase Mx)
2  const divs = document.querySelectorAll('div[class^="M"]');
3
4  let count = 0;
5
6  divs.forEach((div) => {
7    const className = div.className;
8    const match = className.match(/^M(\d+)$/);
9
10   if (match && match[1] !== (divs.length - 1).toString()) {
11     count++;
12   }
13 });
14
15 console.log("Los mensajes cifrados son:", count);
16
```

Figura 4: Código para contar automáticamente los mensajes cifrados

3.2. Muestra la cantidad de mensajes por consola

La implementación del código no solo cuenta la cantidad de mensajes cifrados, sino que también imprime esta cantidad en la consola. Esto se puede observar en la línea 15 de la figura 4, donde se registra el número de mensajes cifrados encontrados. La salida en la consola se muestra en la figura 5.

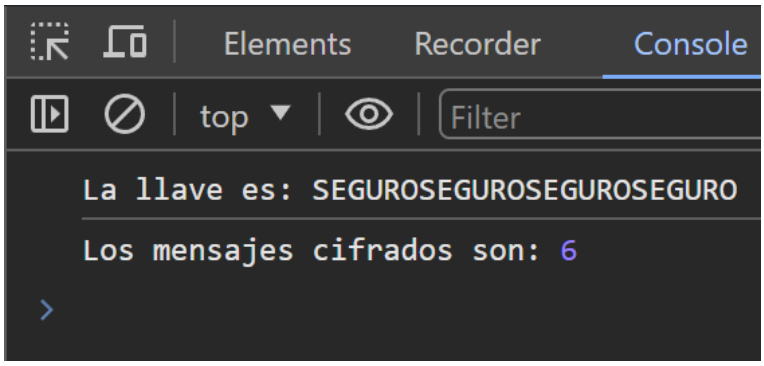

Figura 5: Cantidad de mensajes cifrados impresos en la consola

4. Desarrollo (Parte 3)

4.1. Importa la librería cryptoJS

Para importar la librería de cryptoJS, se realizó una búsqueda en Google y se encontró la fuente en cdnjs de Cloudflare [3]. Se buscó la librería cryptoJS y se copió el enlace de importación desde la página web [4] tal y como se puede observar en la siguiente figura:

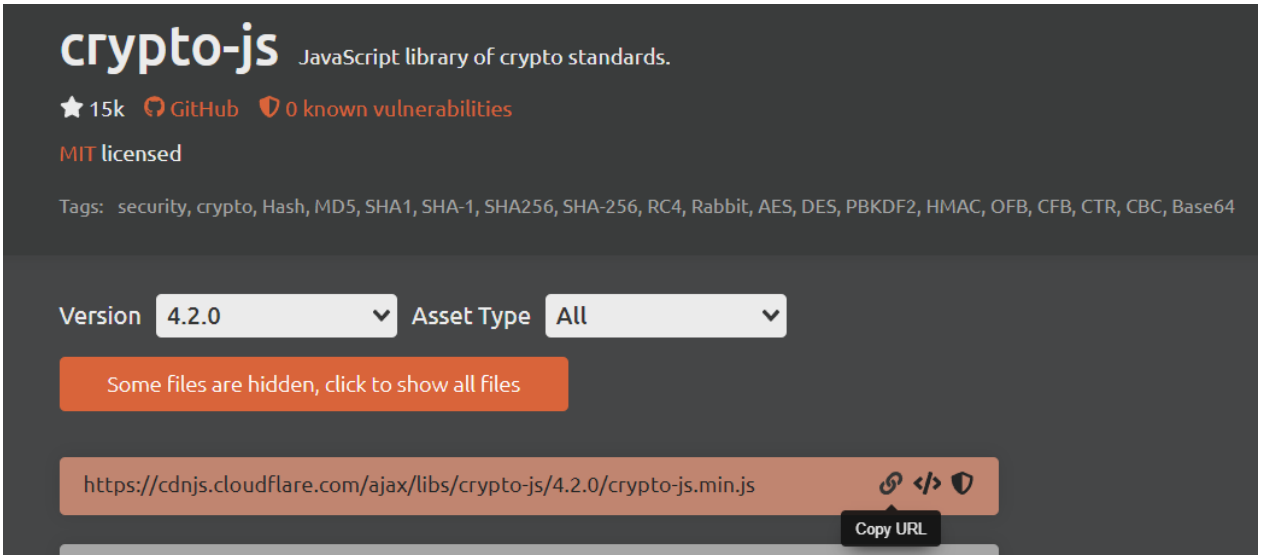

Figura 6: Copia del enlace de importación de la librería cryptoJS desde cdnjs

Con el enlace copiado, se procedió a importar la librería en el script de Tampermonkey, como se muestra en la figura 7.

Figura 7: Importación de la librería cryptoJS en el script de Tampermonkey

4.2. Utiliza SRI en la librería CryptoJS

Para asegurar la integridad de la librería importada, se utilizó Subresource Integrity (SRI). El hash SRI se copió de la página de cdnjs [4].

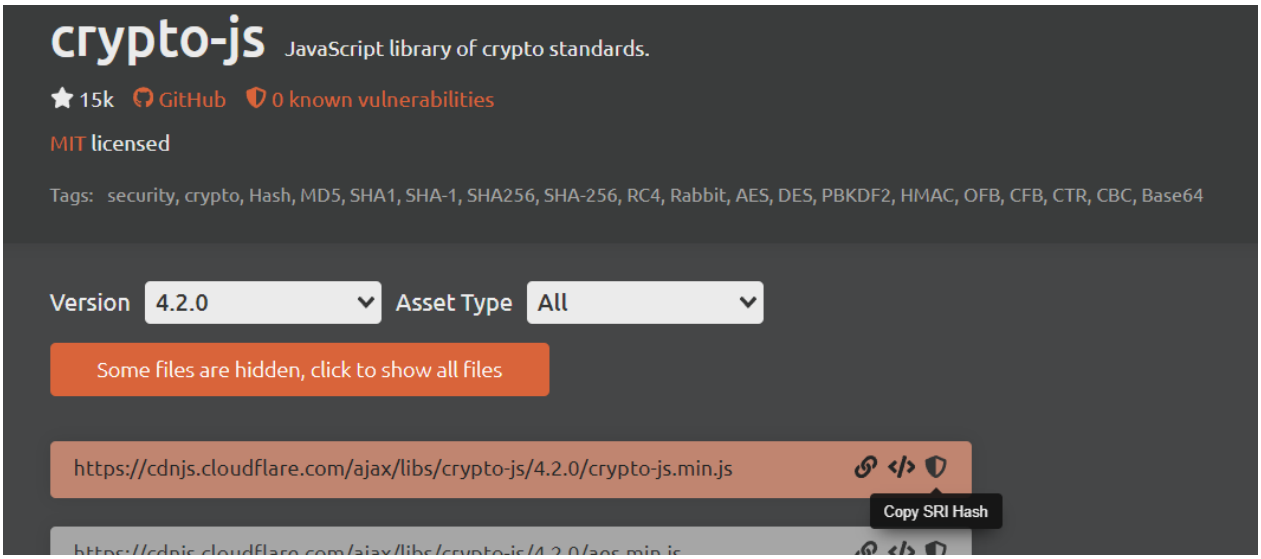

Figura 8:

Para corroborar el uso de SRI, se consultó la documentación de Tampermonkey [5] para comprender la integración de SRI. Siguiendo la documentación, el código quedó de la siguiente manera, como se muestra en la figura 9.

```
1 // @match        https://criptolab04.tiiny.site/
2 // @grant        none
3 // @require      https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.2.0/crypto-js.min.js#
  sha512-a+SUDuwNzXDvz4XrIcXHuCf089/iJAoN4lmrXJg18XnduKK6YlDHNRalv4yd1N40OKI80tFidF+rqTFKGP
  oWFQ==
4 // ==/UserScript==
```

Figura 9: Importación de la librería cryptoJS con SRI en el script de Tampermonkey

4.3. Repercusiones de SRI inválido

Para demostrar las repercusiones de un Subresource Integrity (SRI) inválido, se modificó el hash de integridad en la línea 3 del script, como se muestra en la figura 10. Se cambiaron los primeros caracteres por ÇRIPTOHOLAXD"para asegurar que el hash calculado por Tampermonkey difiera del proporcionado. Esto se hizo con el fin de observar el mensaje de error que se genera en la consola del navegador. El mensaje de error resultante se puede ver en la figura 11.

```
1 // @match        https://criptolab04.tiiny.site/
2 // @grant        none
3 // @require      https://cdnjs.cloudflare.com/ajax/libs/crypto-js/4.2.0/crypto-js.min.js#
  sha512-a+CRIPTOHOLAXDXDIcXHuCf089/iJAoN4lmrXJg18XnduKK6YlDHNRalv4yd1N40OKI80tFidF+rqTFKGP
  oWFQ==
4 // ==/UserScript==
5
```

Figura 10: Modificación del hash SRI para inducir un error

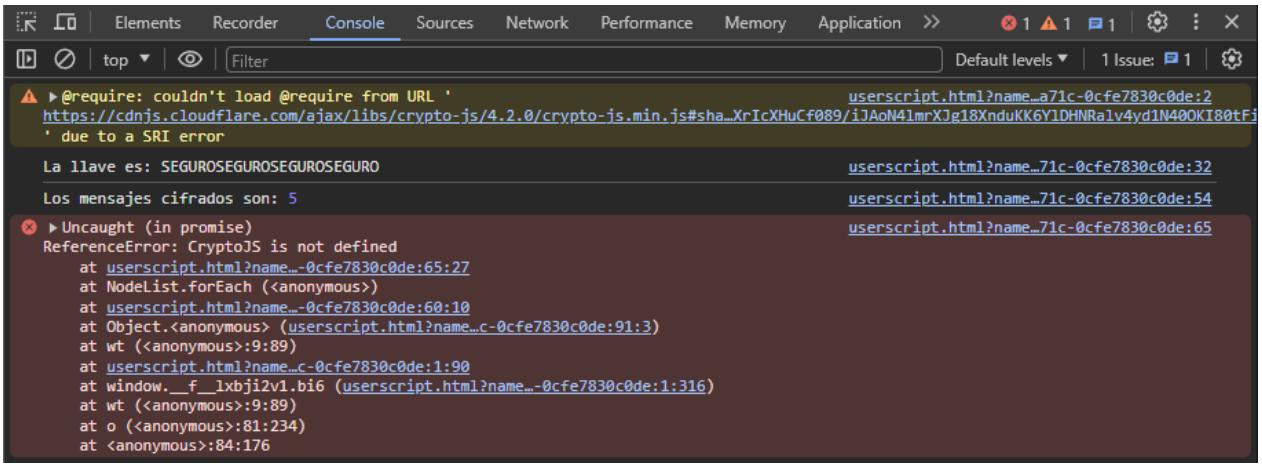

Figura 11: Mensaje de error en la consola del navegador debido a un SRI inválido

4.4. Logra descifrar uno de los mensajes

Como se mencionó al principio de este informe, se intentó descifrar los mensajes utilizando DES con ECB sin obtener resultados satisfactorios. Sin embargo, al emplear Triple DES con ECB, se logró descifrar con éxito. A continuación, se muestra el proceso de descifrado del primer mensaje cifrado, como se observa en la Figura 12.

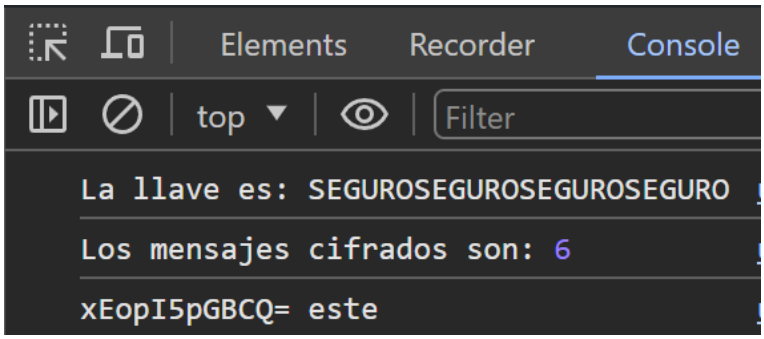

Figura 12: Descifrado del primer mensaje utilizando Triple DES

4.5. Imprime todos los mensajes por consola

Para descifrar e imprimir todos los mensajes cifrados, se implementó el siguiente código que recorre todos los elementos con clase M, los desencripta utilizando Triple DES y el modo de operación ECB, y luego los imprime por consola. Este proceso se puede observar en la figura 13.

```
//Parte 3, descifrar los mensajes
if (key) {
  Let decryptedMessages = "";

  divs.forEach((div) => {
    const encryptedMessage = div.id;

    if (encryptedMessage) {
      //Descifrar el mensaje utilizando 3DES y la clave
      const decrypted = CryptoJS.TripleDES.decrypt(
        {
          ciphertext: CryptoJS.enc.Base64.parse(encryptedMessage),
        },
        CryptoJS.enc.Utf8.parse(key),
        {
          mode: CryptoJS.mode.ECB,
          padding: CryptoJS.pad.Pkcs7,
        }
      );

      const decryptedText = decrypted.toString(CryptoJS.enc.Utf8);

      //Imprime el mensaje cifrado y descifrado por consola
      console.log(`${encryptedMessage} ${decryptedText}`);

      //Agregar el mensaje descifrado al final del documento
      decryptedMessages += `${decryptedText}<br>`;
    }
  });
```

Figura 13: Código para descifrar e imprimir todos los mensajes

El resultado de la impresión por pantalla se puede ver a continuación en la figura 14.

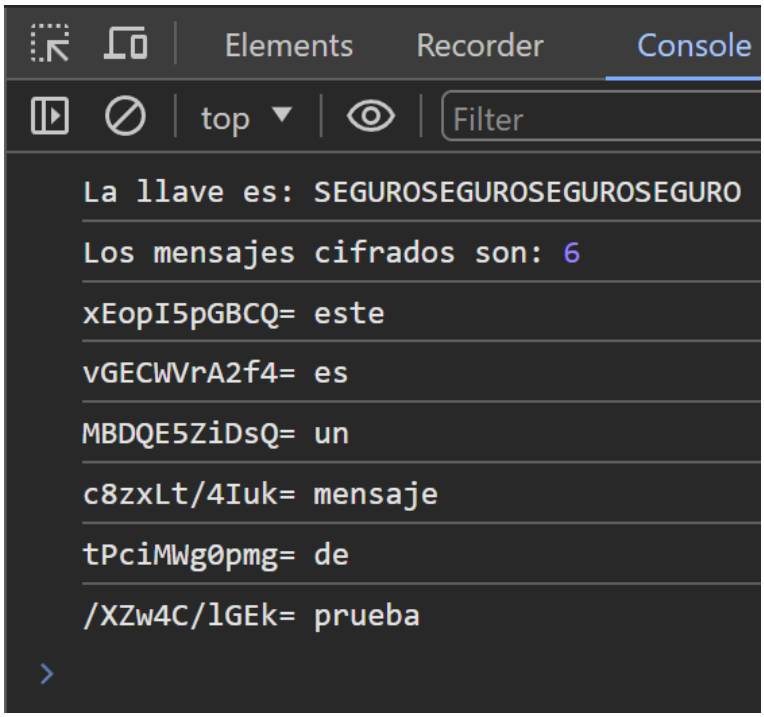

Figura 14: Impresión de todos los mensajes descifrados en la consola

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

Como se observa en la figura 13, línea 27, se agrega al final del documento el mensaje descifrado en texto plano. La creación del div y su adjunción al final del documento se puede observar en la figura 15.

```
1 //Se crea un nuevo elemento div y se agregan los mensajes descifrados en el
  body
2 const decryptedMessagesDiv = document.createElement("div");
3 decryptedMessagesDiv.innerHTML = `${decryptedMessages}`;
4 document.body.appendChild(decryptedMessagesDiv);
```

Figura 15: Creación y adjunción del div con mensajes descifrados

El resultado de lo anterior se puede ver en la figura 16, donde se muestra el documento con los mensajes descifrados agregados.

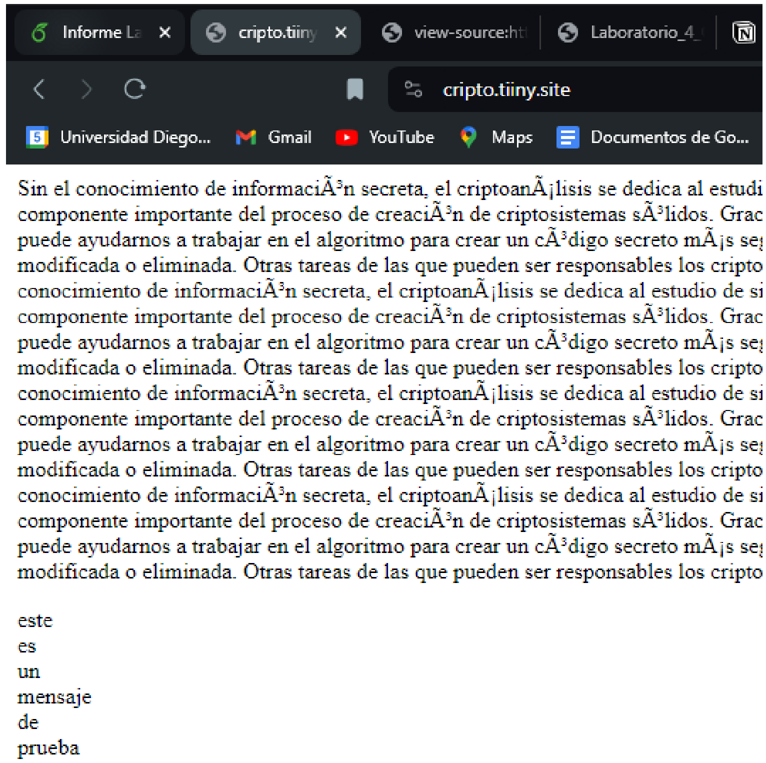

Figura 16: Resultado de los mensajes descifrados agregados al documento

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

Para este apartado, se decidió crear una página similar a la proporcionada en el laboratorio. Esta nueva página tiene una clave distinta y una cantidad diferente de mensajes cifrados. Esta página también se agregó en el match del script, como se puede ver en la línea 1 de la figura 10. A continuación, se muestra la consola al momento de cargar la página web https://criptolab04.tiiny.site/.

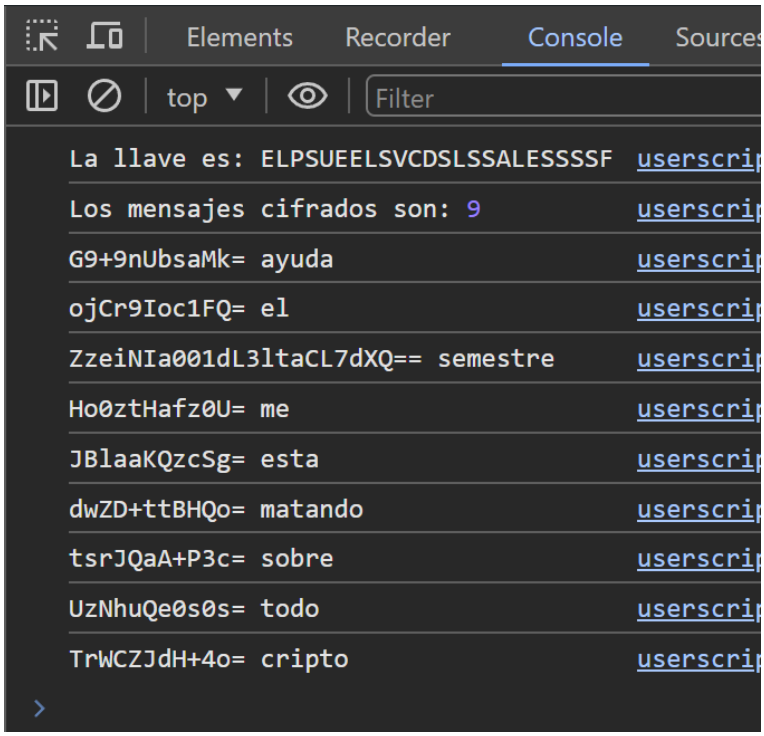

Figura 17: Consola al cargar la página web de prueba

Posteriormente, se puede observar que los mensajes también se agregan en formato de texto plano al final del documento, como se muestra en la figura 18.

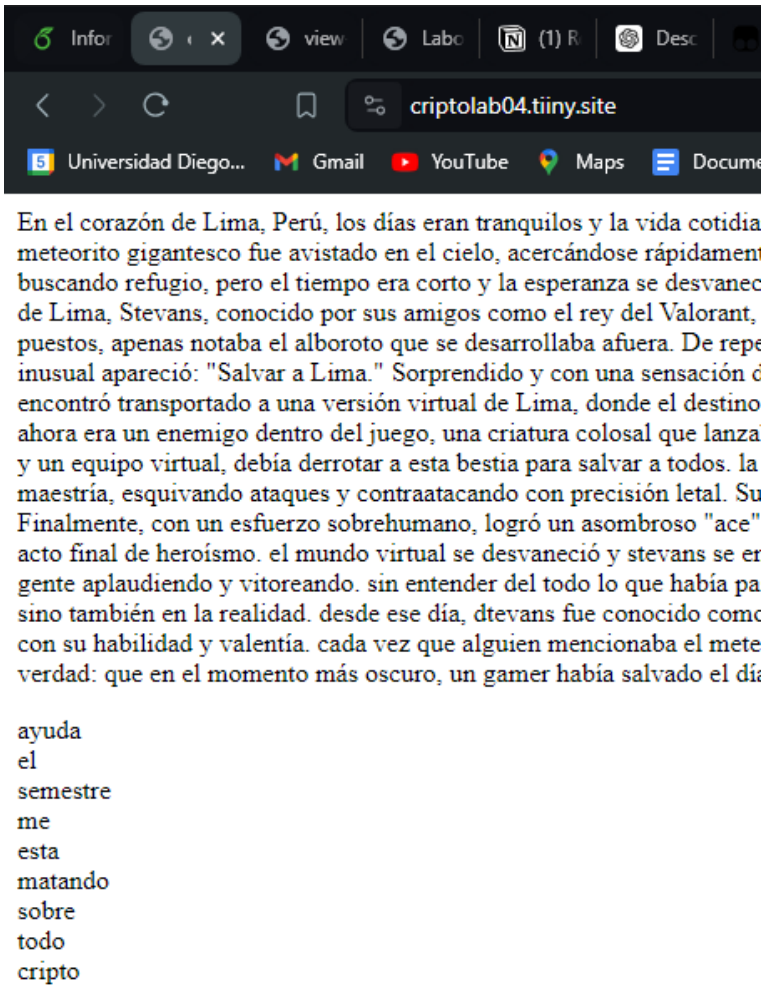

Figura 18: Mensajes descifrados agregados al documento en la página de prueba

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

El código del script.js y el HTML de la página web mencionada se pueden encontrar en el siguiente repositorio de GitHub: https://github.com/cesarmunozr/lab04_cripto.

Conclusiones y comentarios

Para concluir este laboratorio, se pudo demostrar los conocimientos adquiridos en la cátedra de criptografía. A través del criptoanálisis aplicado a la página web, se identificó y extrajo con éxito la clave oculta en las letras mayúsculas del texto. Este análisis, complementado con un enfoque de prueba y error para determinar el algoritmo de cifrado, permitió descifrar los mensajes encriptados utilizando Triple DES en modo ECB.

El uso de herramientas como Tampermonkey y la librería CryptoJS fue fundamental para automatizar el proceso de extracción y descifrado de los mensajes. La implementación del script mostró ser robusta y efectiva, funcionando correctamente incluso cuando se probó con diferentes textos y cantidades de mensajes encriptados.

Además, se demostró la importancia de utilizar Subresource Integrity (SRI) para asegurar la integridad de las librerías importadas, garantizando que no han sido modificadas de manera maliciosa.

El ejercicio también puso de manifiesto la necesidad de un enfoque meticuloso y detallado en el análisis de sistemas criptográficos, resaltando cómo pequeños detalles en la estructura del contenido pueden proporcionar las pistas necesarias para romper un cifrado.

Para finalizar, este laboratorio no solo reforzó los conceptos teóricos aprendidos en clase, sino que también proporcionó una experiencia práctica valiosa en el uso de técnicas de criptoanálisis y la implementación de scripts automatizados para resolver problemas de seguridad criptográfica.

Issues

El primer desafío fue encontrar la clave en el HTML. Inicialmente, se pensó que podría estar oculta en algún div o elemento similar. Tras consultar con compañeros y examinar minuciosamente cada detalle del código HTML, se logró identificar el patrón de las letras mayúsculas como la clave.

Otro problema surgió al intentar determinar el modo de operación del cifrado utilizado. Sin indicios claros en texto plano, se recurrió a un proceso de descarte. Este proceso implicó eliminar posibles modos de operación hasta reducir la lista a los más probables, lo cual llevó finalmente a identificar el modo ECB.

En cuanto al algoritmo de cifrado, la dificultad radicó en discernir cuál era el correcto. Utilizando la documentación disponible y un indicio proporcionado por el profesor sobre la librería CryptoJS, se pudo reducir la lista a dos posibles algoritmos: DES y Triple DES. Las pruebas posteriores confirmaron que Triple DES era el algoritmo utilizado.

Finalmente, se enfrentó el reto de utilizar Subresource Integrity (SRI). Inicialmente, se intentó crear una función para verificar la integridad, pero al buscar en la documentación se descubrió que Tampermonkey gestiona automáticamente esta verificación, lo que simplificó el proceso y evitó la necesidad de escribir código adicional.

Referencias

[1] OpenAI. (s.f.). ChatGPT. Recuperado el 05 de junio de 2024. de `https://openai.com/chatgpt`

[2] CryptoJS. (s.f.). CryptoJS Documentation. Recuperado el 11 de junio de 2024, de `https://cryptojs.gitbook.io/docs`

[3] cdnjs. (s.f.). CryptoJS Library. Recuperado el 11 de junio de 2024, de `https://cdnjs.com/libraries/crypto-js`

[4] cdnjs. (s.f.). CryptoJS Library. Recuperado el 11 de junio de 2024, de `https://cdnjs.com/libraries/crypto-js`

[5] Tampermonkey. (s.f.). Tampermonkey Documentation: Subresource Integrity (SRI). Recuperado el 11 de junio de 2024, de `https://www.tampermonkey.net/documentation.php?locale=en#api:Subresource_Integrity`