

# Proyecto 2

Semana: 09

## Nombre del estudiante:

Damaris Alicia Romero Flores

Milton Alejandro Barrientos Hernández

## Número de cuenta:

32111500

32111619

## Sede de estudio:

CEUTEC Centroamérica

## Docente:

Oscar Fernando D'Cuire Galeano

## Sección:

1009

## Fecha de entrega:

04/12/22

## Contenido

Salt & Pepper .....	3
Salt .....	3
Pepper .....	3
AES .....	5
Funcionamiento .....	5
El proceso de cifrado AES .....	6
Huffman Coding .....	7
LZW .....	9
Instalación.....	10
Bitácora.....	11
Recursos .....	12
Bibliografía.....	13

## Salt & Pepper

### Salt

En criptografía, la **sal** (en inglés, *salt*) comprende bits aleatorios que se usan como una de las entradas en una función derivadora de claves. La otra entrada es habitualmente una contraseña. La salida de la función derivadora de claves se almacena como la versión cifrada de la contraseña. La sal también puede usarse como parte de una clave en un cifrado u otro algoritmo criptográfico. La función de derivación de claves generalmente usa una función hash. A veces se usa como sal el vector de inicialización, un valor generado previamente.

Los datos con sal complican los ataques de diccionario que cifran cada una de las entradas del mismo: cada bit de sal duplica la cantidad de almacenamiento y computación requeridas.

Para mayor seguridad, el valor de sal se guarda en secreto, separado de la base de datos de contraseñas. Esto aporta una gran ventaja cuando la base de datos es robada, pero la sal no. Para determinar una contraseña a partir de un hash robado, el atacante no puede simplemente probar contraseñas comunes (como palabras del idioma inglés o nombres), sino calcular los hashes de caracteres aleatorios (al menos la porción de la entrada que se sabe es la sal), lo que es mucho más lento.

### Pepper

En Cifrado, un pepper es una secuencia secreta de bits añadidos a una entrada, por ejemplo, una contraseña, antes de hash con una función hash criptográfica. La pimienta tiene una función similar a la sal, pero a diferencia de esta última no se guarda junto con la salida de la función hash. Las configuraciones más comunes para pepper son las siguientes: pepper aumenta la seguridad de la base de datos en la que se guardan los valores de salida de la función hash, ya que aumenta el número de elementos que necesita tener para recuperar la entrada, lo que dificulta un ataque de fuerza bruta.

A continuación, se muestra un ejemplo incompleto sobre el uso de una pimienta constante para guardar contraseñas. La siguiente tabla contiene dos combinaciones de nombre de usuario y contraseña. La contraseña no se guarda, y el pepper 44534c70c6883de2 de 8 bytes (64 bits) se guarda en un lugar seguro separado de los valores de salida de hash. A diferencia de salt, pepper no proporciona protección a los usuarios que usan la misma contraseña, pero protege contra ataques de diccionario, a menos que el atacante tenga el valor pepper disponible. Debido a que el mismo pepper no se comparte entre diferentes aplicaciones, Un atacante no puede reutilizar hashes de una base de datos comprometida a otra. Un esquema completo de ahorro de contraseña generalmente incluye el uso de sal y pimienta.

## AES

Actualmente, hay tres tipos de cifrado AES: 128 bits, 192 bits y 256 bits, donde este último por su longitud en el número de bits es el más seguro. Esto se diseñó basándose en la **Ley de Moore**, ya que las primeras pruebas demostraron que, en un tiempo relativamente corto, la potencia de los procesadores podría romper el cifrado más débil y, por tanto, con menor número de bits en periodos de tiempo cada vez más bajo.

Por ello, aunque este tipo de encriptación empezó con 56 bits de longitud, rápidamente se cambió a los tres tipos nombrados para hacerlo más seguro. Advanced Encryption Standard o AES tiene una ventaja bastante clara frente a otros competidores: tiene una naturaleza abierta, lo cual significa que se puede usar tanto en lo público como en lo privado, sea para fines comerciales o no.

Además, es un sistema de clave simétrica, lo cual le otorga una mayor seguridad, ya que la clave usada debe ser conocida para el cifrado como para el descifrado y tanto emisor como receptor necesitan una copia de la llave maestra correspondiente. Esto tiene una importante ventaja en el rendimiento de un sistema de cifrado, puesto que como se obliga a remitente y destinatario a saber la clave, se requiere menos potencia computacional para hacer viajar la información, puesto que siempre va segura. Esto es lo que nos asegura que un archivo solo pueda ser abierto por el mismo sistema que ha codificado los datos en AES-256.

### Funcionamiento

Para entender el funcionamiento del cifrado AES hemos de entender que la información original sufre una transformación, donde los significantes binarios son modificados de tal manera que sin el decodificador pertinente no se pueden entender. Entiéndase significativo como el código binario que codifica lo que son datos e instrucciones que el procesador ha de ejecutar. Así pues, de la misma manera que un conjunto de letras y números ilegibles para nosotros no tiene sentido alguno, para un procesador tampoco.

## El proceso de cifrado AES

Cada byte de datos se sustituye por otro mediante una tabla predeterminada para a continuación coger cada matriz 4×4 y moverla de la siguiente manera:

- Los bytes de la segunda fila se mueven un espacio a la izquierda-
- Los bytes en la tercera fila se mueven dos espacios.
- En cuanto a los de la cuarta fila, estos se mueven tres espacios

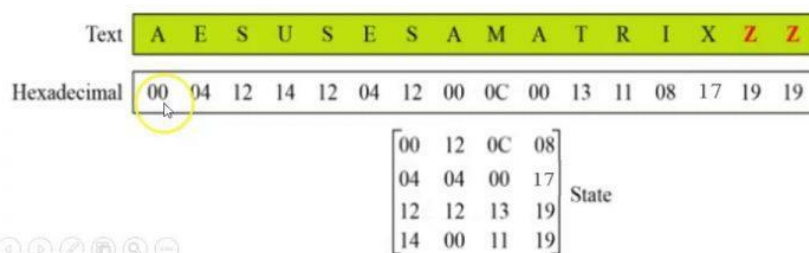
Finalmente, se mezclan las columnas y se añade la clave inicial al conjunto y vuelta a empezar.

Esto genera un texto cifrado que nada tiene que ver con el original y que para descifrarlo se requiere hacer los pasos inversos conociendo previamente dicha clave.

## AES (Advanced Encryption Standard)

### □ Plain Text transform into Matrix Form

- For Example, "AES USES A MATRIX".
- Plain text (128-bit) converts into 4x4 matrix of bytes.



	DEC	HEX		DEC	HEX
A	00	00	N	13	0D
B	01	01	O	14	0E
C	02	02	P	15	0F
D	03	03	Q	16	10
E	04	04	R	17	11
F	05	05	S	18	12
G	06	06	T	19	13
H	07	07	U	20	14
I	08	08	V	21	15
J	09	09	W	22	16
K	10	0A	X	23	17
L	11	0B	Y	24	18
M	12	0C	Z	25	19

## Huffman Coding

El algoritmo de Huffman se usa para la compresión o encriptación de datos mediante el estudio de la frecuencia de aparición de caracteres. Fue desarrollado por el norteamericano **David Albert Huffman** en 1952 mientras hacía el doctorado en el MIT. El método fue publicado en una revista como *A Method for the Construction of Minimum-Redundancy Codes*.

El algoritmo funciona a partir de un conjunto dado de símbolos con sus respectivos pesos. Los pesos son la frecuencia de aparición en una cadena. Por ejemplo, en la cadena *Genciencia* el peso del símbolo *i* es 2, ya que aparece en dos ocasiones. La salida del algoritmo es el mismo conjunto de símbolos de entrada codificado mediante un código binario con un tamaño menor.

La longitud de cada código no es idéntica para todos los símbolos: se asignan códigos cortos a los símbolos utilizados con más frecuencia (los que aparecen más a menudo), mientras que los símbolos menos frecuentes reciben códigos binarios más largos. La expresión **Código de Longitud Variable** (VLC) se utiliza para indicar este tipo de código porque ningún código es el prefijo de otro. De este modo, la sucesión final de códigos con longitudes variables será en promedio más pequeña que la obtenida con códigos de longitudes constantes.

El codificador Huffman crea una estructura arbórea ordenada con todos los símbolos y la frecuencia con que aparecen. Las ramas se construyen en forma recursiva comenzando con los símbolos menos frecuentes.

La construcción del árbol se realiza ordenando en primer lugar los símbolos según la frecuencia de aparición. Los dos símbolos con menor frecuencia de aparición se eliminan sucesivamente de la lista y se conectan a un nodo cuyo peso es igual a la suma de la frecuencia de los dos símbolos. El símbolo con menor peso es asignado a la rama 1, el otro a la rama 0 y así sucesivamente, considerando cada nodo formado como un símbolo nuevo, hasta que se obtiene un nodo principal  
llamado *raíz*.

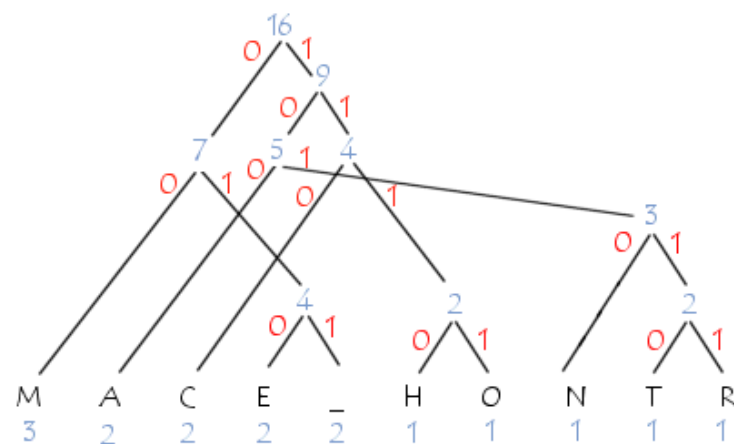
El código de cada símbolo corresponde a la sucesión de códigos en el

camino, comenzando desde este carácter hasta la raíz. De esta manera, cuanto más dentro del árbol esté el símbolo, más largo será el código.

Analicemos la siguiente oración: "COMMENT\_CA\_MARCHE". Las siguientes son las frecuencias de aparición de las letras:

M	A	C	E	_	H	O	N	T	R
3	2	2	2	2	1	1	1	1	1

Éste es el árbol correspondiente:



Los códigos correspondientes a cada carácter son tales que los códigos para los caracteres más frecuentes son cortos y los correspondientes a los símbolos menos frecuentes son largos:

M	A	C	E	_	H	O	N	T	R
00	100	110	010	011	1110	1111	1010	10110	10111

Las compresiones basadas en este tipo de código producen buenas proporciones de compresión, en particular, para las imágenes monocromáticas (faxes, por ejemplo). Se utiliza especialmente en las recomendaciones T4 y T5 utilizadas en ITU-T.



## LZW

LZW es un algoritmo muy rápido tanto para la compresión como para la descompresión, basado en la multiplicidad de aparición de secuencias de caracteres en la cadena que se debe codificar. Su principio consiste en sustituir patrones con un código de índice y construir progresivamente un diccionario.

Además, funciona en [bits](#) y no en [bytes](#), por lo tanto, no depende de la manera en que el procesador codifica información. Es uno de los algoritmos más populares y se utiliza particularmente en formatos TIFF y GIF. Dado que el método de compresión LZW ha sido patentado por Unisys, el que se utiliza en imágenes PNG es el algoritmo LZ77, por el que no se pagan derechos de autor.

El diccionario comienza con los 256 valores de la tabla [ASCII](#). El archivo a comprimir se divide en cadenas de bytes (por lo tanto, para las imágenes monocromáticas codificadas en 1 bit, esta compresión no es muy eficaz), cada una de estas cadenas se compara con el diccionario y se agrega si no se encuentra ahí.

### **Compresión**

El algoritmo pasa por la cadena de información y la codifica. Si una cadena nunca es más corta que la palabra más larga del diccionario, ésta se transmite.

### **Descompresión**

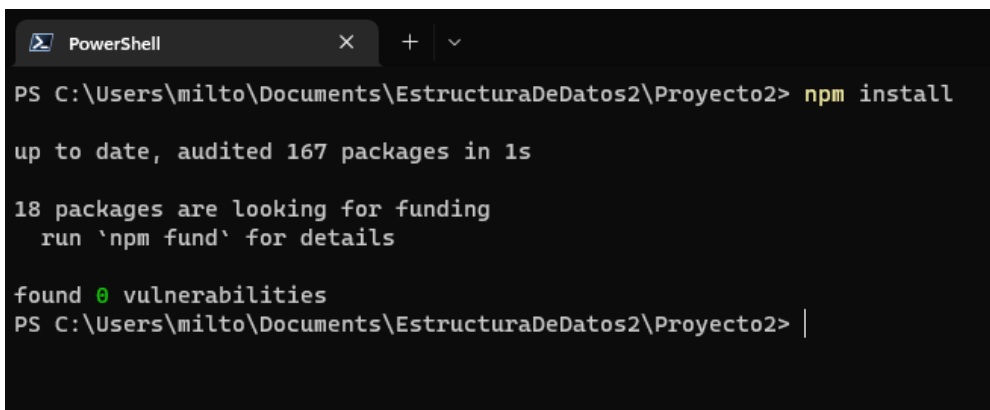
Durante la descompresión, el algoritmo reconstruye el diccionario en la dirección opuesta; por lo tanto, no necesita almacenarse.

## Instalación

El proyecto funciona bajo el motor NodeJS, así que debemos tenerlo instalado en nuestra computadora.

### [NodeJS – Página Oficial](#)

1. Clonar el repositorio
2. Abrir la terminal, dirigirnos a la carpeta donde tenemos nuestro proyecto.
3. Ejecutamos el comando “npm install”, este será el encargado de instalar todos los módulos necesarios para el funcionamiento de nuestro API.



```
PowerShell
PS C:\Users\milto\Documents\EstructuraDeDatos2\Proyecto2> npm install

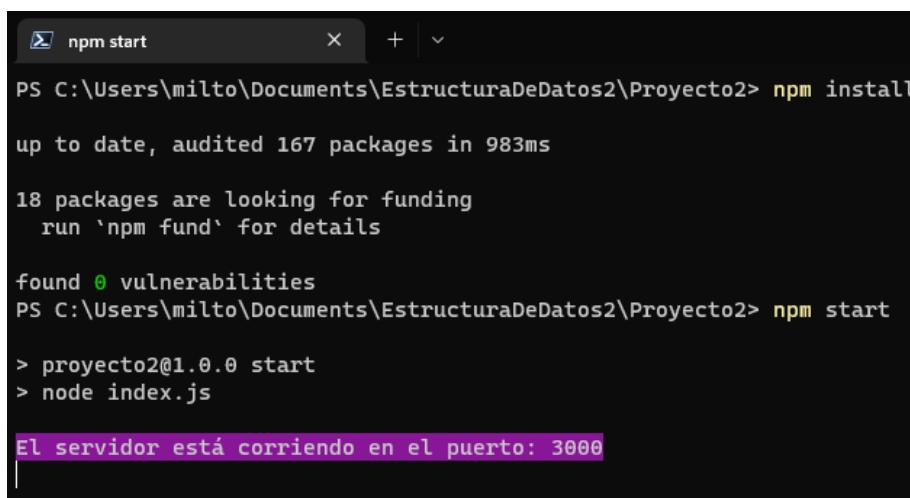
up to date, audited 167 packages in 1s

18 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
PS C:\Users\milto\Documents\EstructuraDeDatos2\Proyecto2> |
```

En el momento en el que veamos que todo fue instalado, podemos proseguir con el siguiente paso.

4. Ejecutar el comando “npm start” para iniciar el API



```
npm start
PS C:\Users\milto\Documents\EstructuraDeDatos2\Proyecto2> npm install

up to date, audited 167 packages in 983ms

18 packages are looking for funding
  run 'npm fund' for details

found 0 vulnerabilities
PS C:\Users\milto\Documents\EstructuraDeDatos2\Proyecto2> npm start

> proyecto2@1.0.0 start
> node index.js

El servidor está corriendo en el puerto: 3000
```

Al ver ese mensaje, el API ya estará corriendo y listo para recibir peticiones http en el host de

nuestra PC y puerto 3000.

### Bitácora

Actividad	Fecha	Observaciones
Crear el repositorio en Git	26/11/2022	Aquí empezaría todo el desarrollo
Investigación sobre varios métodos	26/11/2022	Hay una gran cantidad de métodos, así que decidimos implementar los que logramos encontrar con mayor facilidad para Javascript
Diseño HTML	27/11/2022	Creamos un ligero diseño en HTML y CSS para la presentación del mismo.
Cifrado Salt y AES	27/11/2022	Aquí implementamos los dos métodos de cifrado.
Solución de problema con cifrado AES	29/11/2022	Después de implementar, nos dimos cuenta que el método AES estaba bastante inestable, así que estuvimos dos días resolviendo los problemas de inestabilidad.
Compresión Huffman y LWZ	01/12/2022	Implementamos ambas compresiones y funcionaron de maravilla a la primera.
Descanso	02/12/2022	Descancito porque postergaron la fecha (•'∪'•)
Grabación del Video	04/12/2022	Grabamos todos los videos necesarios para la presentación el proyecto.

## Recursos

Enlace al video de youtube: <https://youtu.be/grXvShzDiZI>

Enlace al repositorio: <https://github.com/crywhat7/proyecto2EDD2.git>

## Bibliografía

Jiménez, A. (2006, 26 octubre). *Algoritmo de Huffman*. Xataka Ciencia.

<https://www.xatakaciencia.com/matematicas/algoritmo-de-huffman>

*Salt (cifrado) - Implementaciones en aplicaciones web, Implementaciones De Unix,*

*Beneficio, Errores comunes, Ejemplo de uso | KripKit.* (s. f.).

<https://kripkit.com/salt-cifrado/>

seyi. (2020, 10 agosto). *¿Cómo funciona la compresión de archivos? | Codificación*

*Huffman*. YouTube. <https://www.youtube.com/watch?v=85NFcU8yTSs>