

Instituto Politécnico Nacional

Unidad Profesional Interdisciplinaria   
de   
Ingeniería Y Ciencias Sociales Y Administrativas

Materia: Seguridad Informática

Alumnos:

Estrada Pérez Hisao

Mecca Paredes Franco Samuel

Ríos Crisóstomo Omar

Ciclo: 2020/1

Profesor: Sandoval Gonzalez Victor L.

Secuencia: 3NM60

Contenido

[Introducción 2](#_Toc23965835)

[Algoritmos HASH 3](#_Toc23965836)

[Propiedades 3](#_Toc23965837)

[Funcionamiento 7](#_Toc23965838)

[Diseño 10](#_Toc23965839)

[Tokens o llaves 10](#_Toc23965840)

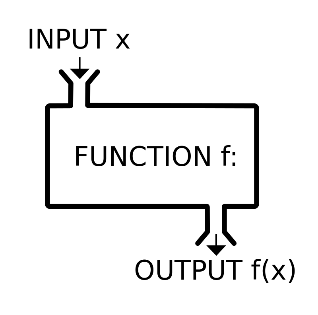
[Operaciones XOR, AND, OR y NOT 10](#_Toc23965841)

[Conclusión 12](#_Toc23965842)

[Referencias 13](#_Toc23965843)

# Introducción

En matemáticas, una función es una relación entre conjuntos que se asocia a cada elemento de un primer conjunto para exactamente un elemento del segundo conjunto.



Una función hash es cualquier función que se puede utilizar para asignar datos de tamaño arbitrario a valores de tamaño fijo. Los valores devueltos por una función hash se llaman valores hash, códigos hash, digests, o simplemente hashes. Los valores se usan para indexar una tabla de tamaño fijo llamada tabla hash. El uso de una función hash para indexar una tabla hash se denomina hashing o direccionamiento de almacenamiento de dispersión.

El uso de funciones hash se basa en las propiedades estadísticas de la interacción de teclas y funciones: el comportamiento en el peor de los casos es intolerablemente malo con una probabilidad muy pequeña, y el comportamiento promedio del caso puede ser casi óptimo (colisiones mínimas).

El presente documento mostrará cómo funcionan algunos de los algoritmos más seguros para encriptar información, y procurará la implementación de uno en un lenguaje de programación especifico, el cual puede ser traducido a otro.

# Algoritmos HASH

Una función hash criptográfica (CHF) es una función hash que es adecuada para su uso en criptografía. Es un algoritmo matemático que asigna datos de tamaño arbitrario a una cadena de bits de un tamaño fijo y es unidireccional, es decir, una función que es prácticamente imposible de invertir.

Idealmente, la única forma de encontrar un mensaje que produzca un hash dado es intentar una búsqueda de fuerza bruta de posibles entradas para ver si producen una coincidencia, o usar una tabla de arcoíris de hashes emparejados. Las funciones hash criptográficas son una herramienta básica de la criptografía moderna.

## Propiedades

La función hash criptográfica ideal tiene las siguientes propiedades principales:

Uniforme: Cada valor hash en el rango de salida debe generarse con aproximadamente la misma probabilidad (La razón de este último requisito es que el costo de los métodos basados ​​en hash aumenta considerablemente a medida que aumenta el número de colisiones).

Probable y Medible: Cuando se prueba una función hash, la uniformidad de la distribución de valores de hash puede ser evaluada por la prueba de x2. Esta prueba es una medida de ajuste: es la distribución real de artículos versus la distribución esperada (o uniforme) de artículos.

Las funciones de hash pueden tener algunas propiedades técnicas que hacen que sea más probable que tengan una distribución uniforme cuando se aplican. Uno es el criterio estricto de avalancha: cada vez que se complementa un solo bit de entrada, cada uno de los bits de salida cambia con una probabilidad del 50%. La razón de esta propiedad es que los subconjuntos seleccionados del espacio clave pueden tener baja variabilidad. Para que la salida se distribuya uniformemente, una baja cantidad de variabilidad, incluso un bit, debería traducirse en una gran cantidad de variabilidad (es decir, distribución sobre el espacio de la tabla) en la salida.

Eficiente: En las aplicaciones de almacenamiento y recuperación de datos, el uso de una función hash es una compensación entre el tiempo de búsqueda y el espacio de almacenamiento de datos. Si el tiempo de búsqueda fuera ilimitado, una lista lineal desordenada muy compacta sería el mejor medio; Si el espacio de almacenamiento fuera ilimitado, una estructura accesible al azar indexable por el valor de la clave sería muy grande, muy escasa, pero muy rápida. Una función hash toma una cantidad de tiempo finita para asignar un espacio clave potencialmente grande a una cantidad factible de espacio de almacenamiento que se puede buscar en una cantidad limitada de tiempo, independientemente de la cantidad de claves. En la mayoría de las aplicaciones, es altamente deseable que la función hash sea computable con latencia mínima y secundariamente con un número mínimo de instrucciones.

Universal: Un esquema de hash universal es un algoritmo aleatorio que selecciona una función de hash h entre una familia de tales funciones, de tal manera que la probabilidad de colisión de dos teclas distintas es 1/m, donde m es el número de valores de hash distintos deseado, independientemente de las dos teclas. El hash universal asegura (en un sentido probabilístico) que la aplicación de la función hash se comportará tan bien como si estuviera usando una función aleatoria, para cualquier distribución de los datos de entrada. Sin embargo, tendrá más colisiones que el hashing perfecto y puede requerir más operaciones que una función hash de propósito especial.

Aplicable: Una función hash debería ser aplicable a todas las situaciones en las que se podría usar una función hash. Una función hash que permite solo ciertos tamaños de tabla, cadenas de hasta una cierta longitud, o no puede aceptar una semilla (es decir, permitir doble hashing) no es tan útil como una que sí lo hace.

Determinista: Debe ser una función de los datos que se van a codificar, en el sentido matemático del término.

Rango: A menudo es deseable que la salida de una función hash tenga un tamaño fijo. Pero, si, por ejemplo, la salida está restringida a valores enteros de 32 bits, los valores hash se pueden usar para indexar en una matriz. Tal hashing se usa comúnmente para acelerar las búsquedas de datos

En muchas aplicaciones, el rango de valores hash puede ser diferente para cada ejecución del programa, o puede cambiar a lo largo de la misma ejecución (por ejemplo, cuando se necesita expandir una tabla hash). En esas situaciones, se necesita una función hash que tome dos parámetros: los datos de entrada z y el número n de valores hash permitidos.

Cuando se utiliza este enfoque, se debe elegir la función hash para que el resultado tenga una distribución bastante uniforme entre 0 y n - 1, para cualquier valor de n que pueda ocurrir en la aplicación. Dependiendo de la función, el resto puede ser uniforme solo para ciertos valores de n, por ejemplo, números primos o impares.

Normalizable: En algunas aplicaciones, los datos de entrada pueden contener características que son irrelevantes para fines de comparación. Por ejemplo, al buscar un nombre personal, puede ser conveniente ignorar la distinción entre mayúsculas y minúsculas. Para tales datos, uno debe usar una función hash que sea compatible con el criterio de equivalencia de datos que se usa: es decir, cualesquiera dos entradas que se consideren equivalentes deben producir el mismo valor hash. Esto se puede lograr normalizando la entrada antes de aplicar el hash, como mayúsculas y minúsculas en todas las letras.

Resistencia previa a la Imagen: Dado un valor hash h, debería ser difícil encontrar cualquier mensaje m tal que h = hash (m). Este concepto está relacionado con el de una función unidireccional. Las funciones que carecen de esta propiedad son vulnerables a los ataques de preimagen.

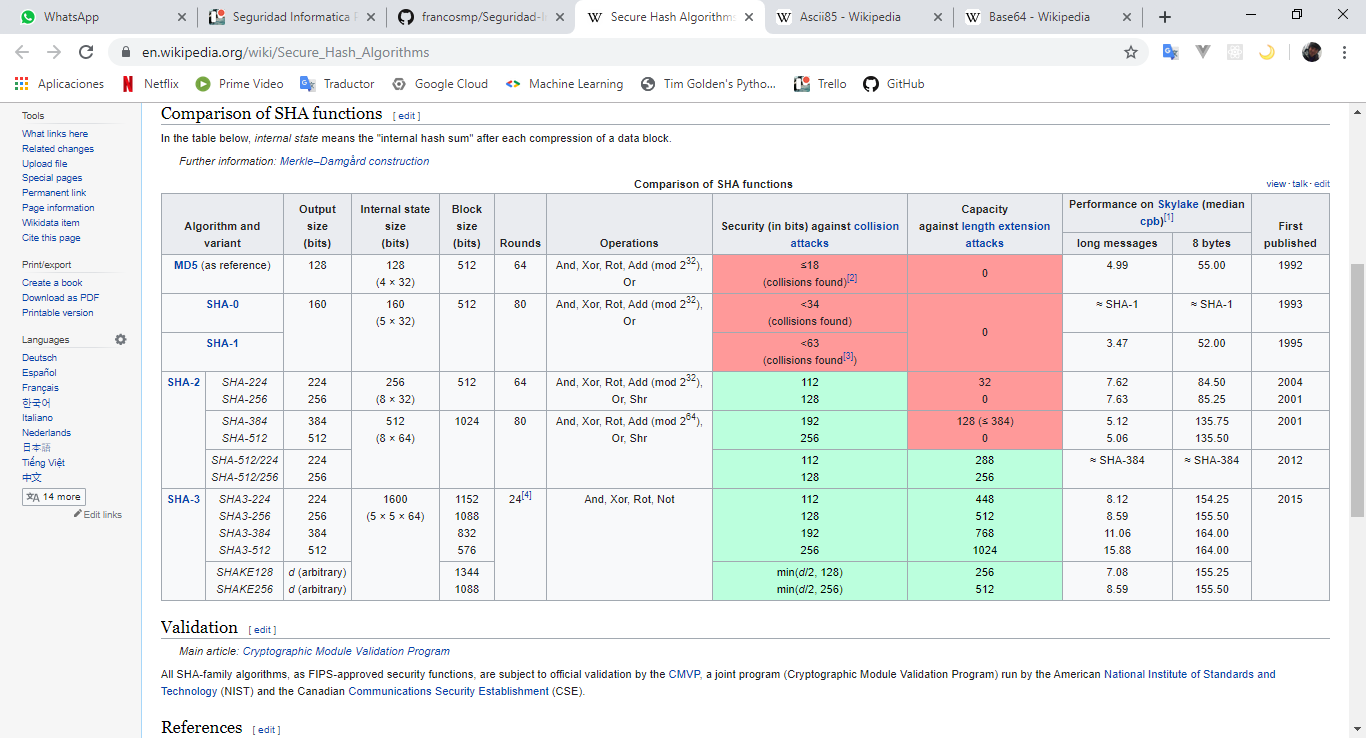
Segunda Resistencia previa a la Imagen: Dada una entrada m1, debería ser difícil encontrar una entrada diferente m2 tal que hash (m1) = hash (m2). Las funciones que carecen de esta propiedad son vulnerables a los ataques de segunda imagen previa.

Resistencia a la colisión: Debería ser difícil encontrar dos mensajes diferentes m1 y m2 de modo que hash (m1) = hash (m2). Tal par se llama colisión hash criptográfica. Esta propiedad a veces se conoce como fuerte resistencia a la colisión. Requiere un valor hash al menos el doble del requerido para la resistencia previa a la imagen; de lo contrario, se pueden encontrar colisiones por un ataque de cumpleaños.

Grado de Dificultad: En la práctica criptográfica, "difícil" generalmente significa "casi con certeza más allá del alcance de cualquier adversario a quien se debe evitar que rompa el sistema mientras se considere importante la seguridad del sistema". Por lo tanto, el significado del término depende en cierta medida de la aplicación, ya que el esfuerzo que un agente malintencionado puede realizar en la tarea suele ser proporcional a su ganancia esperada. Sin embargo, dado que el esfuerzo necesario generalmente se multiplica con la longitud del resumen, incluso una ventaja de mil veces en la potencia de procesamiento se puede neutralizar agregando unas pocas docenas de bits a este último.

## Funcionamiento

Comparación de funciones SHA



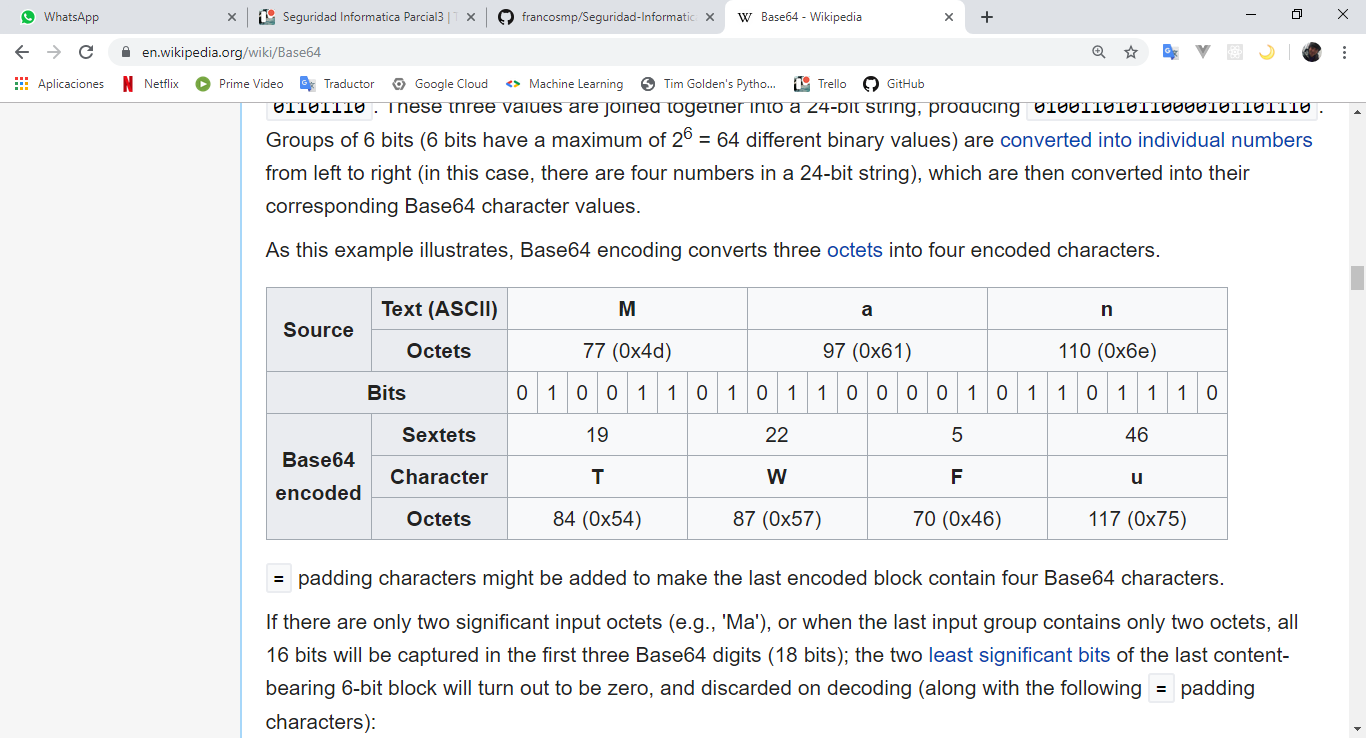
Para ilustrar mejor, utilizaremos la codificación conocida como base 64, que es un grupo de esquemas de codificación de binario a texto que representan datos binarios en un formato de cadena ASCII traduciéndolos a una representación radix-64. El término Base64 se origina a partir de una codificación de transferencia de contenido MIME específica.

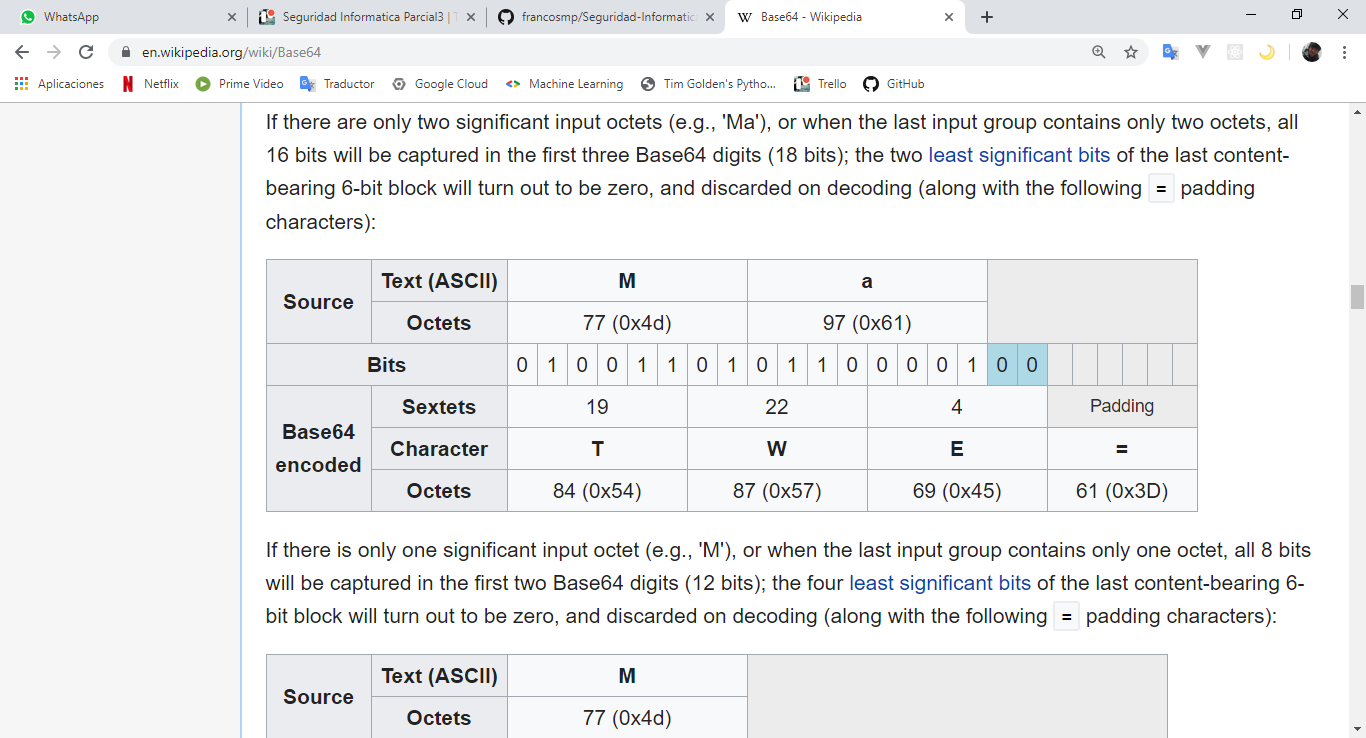
Cada dígito Base64 representa exactamente 6 bits de datos. Por lo tanto, tres bytes de 8 bits (es decir, un total de 24 bits) se pueden representar con cuatro dígitos Base64 de 6 bits.

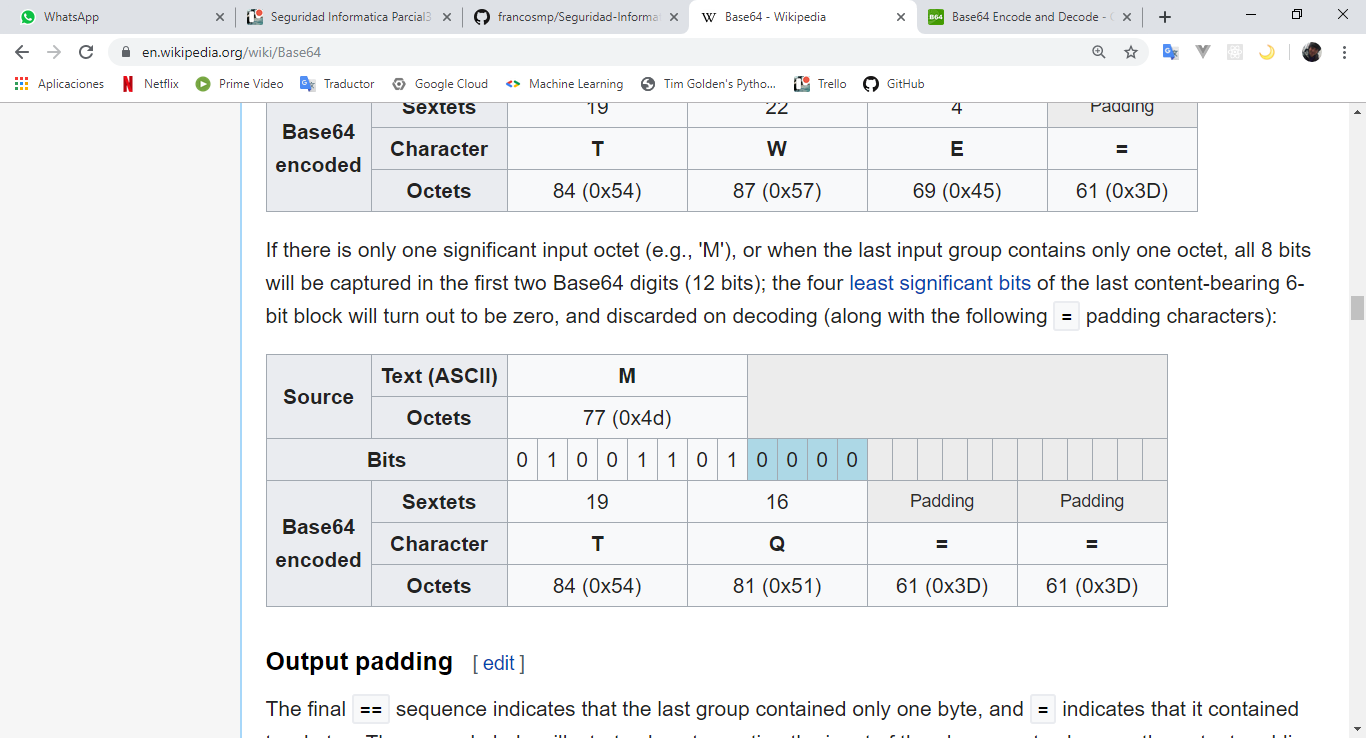
Para homologar algunas características al contenido de las funciones SHA, tendremos:

* Output: variable
* Internal State Size (entran x salen): 8 x 6
* Block Size (capacidad): variable

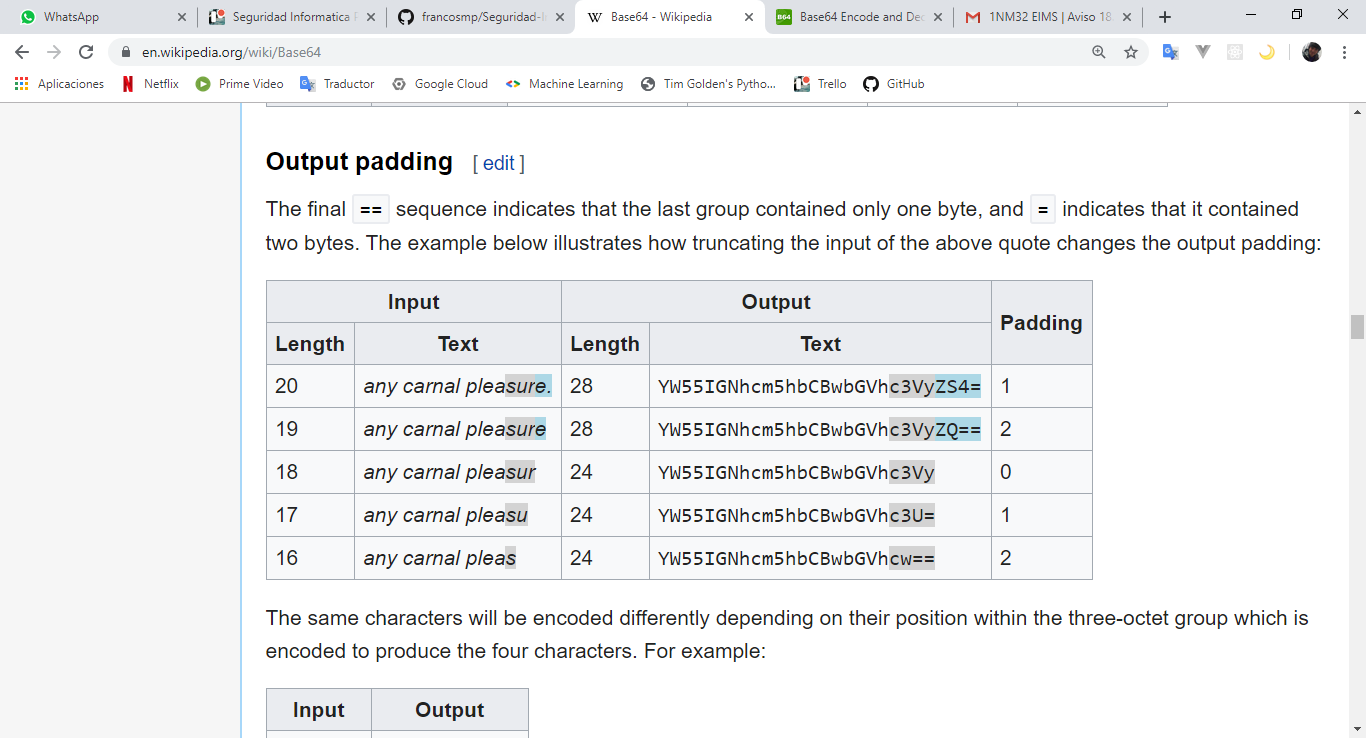
Los siguientes ejemplos ilustran la codificación con base 64.



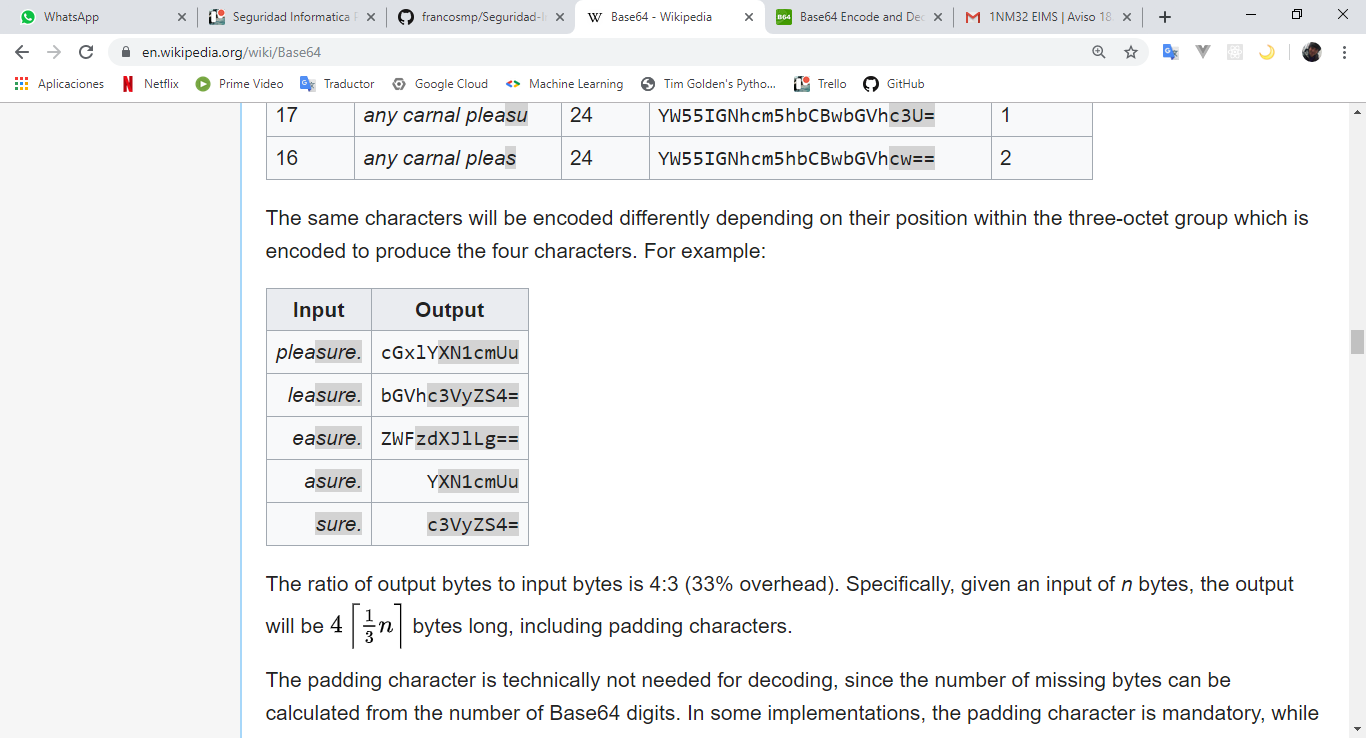




Relleno en base 64.



Los mismos caracteres se codificarán de manera diferente dependiendo de su posición dentro del grupo de tres octetos que se codifica para producir los cuatro caracteres. Por ejemplo:



# Diseño

Para diseñar un algoritmo es necesario conocer algunos conceptos, y funciones de operación de los algoritmos tradicionales. Ya que la mayoría de algoritmos se basan en funciones matemáticas de alta complejidad, trataremos de explicar algunos de la manera más simple posible.

## Tokens o llaves

Algunos algoritmos emplean identificadores o tokens para seguir comportamientos previamente establecidos y descifrar datos. Por lo general, esos tokens poseen características propias de algún elemento al que se relacionen datos a descifrar.

Por ejemplo, para un usuario con identificación “usuario”, se puede tomar como token de autenticación las letras “usr”, de esta manera, cada que el usuario “usuario” realice operaciones que necesiten ser codificadas, se puede usar el token “usr” para crear un flujo de codificación.

## Operaciones XOR, AND, OR y NOT

Para poder codificar, algunos algoritmos utilizan las operaciones XOR, AND, OR y NOT para modificar ciertas características de los datos a codificar. Cabe resaltar que estas operaciones funcionan más rápidamente que funciones definidas a nivel alto de programación ya que tienen relación con las operaciones del procesador.

Por ejemplo, si queremos encriptar la palabra “usuario”, y aplicarle el comando XOR con el token “usr”, tendríamos:

usuario = 01110101 01110011 01110101 01100001 01110010 01101001 01101111

usr = 01110101 01110011 01110010

y al aplicar el comando xor, obtendríamos

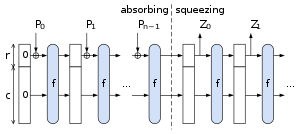
???: = 1110101011100110111010101100001000001110001101000011101

De esta manera, si queremos recuperar la información con el token “usr”, tendríamos que operar “???:” con el token “usr” para obtener la palabra “usuario”.

## Implementaciones en Hardware

Los diseños actuales de funciones de seguridad están abarcando hardware debido a su característica actual de poseer mayor capacidad de operar, de esta manera seria más difícil poder corromper sistemas.

A continuación, se muestra el esquema de una de las funciones mas potentes que operan en hardware, la función KeccaK, y se deja a libre investigación.



# Conclusión

# Referencias

ASCII. (s.f.). Obtenido de Wikipedia: https://es.wikipedia.org/wiki/ASCII

Ascii85. (s.f.). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Ascii85

Base64. (s.f.). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Base64

Birthday\_attack. (s.f.). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Birthday\_attack

Chi\_Squared. (s.f.). Obtenido de https://en.wikipedia.org/wiki/Chi-squared\_test

Collision. (s.f.). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Collision\_(computer\_science)

Cryptographic\_Hash\_Function. (s.f.). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Cryptographic\_hash\_function

Function\_(mathematics). (s.f.). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Function\_(mathematics)

HASH\_Function. (s.f.). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Hash\_function

Knuth, D. E. (1973). *The Art of Computer Programming.* Obtenido de http://broiler.astrometry.net/~kilian/The\_Art\_of\_Computer\_Programming%20-%20Vol%201.pdf

Multipurpose\_Internet\_Mail\_Extensions. (s.f.). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/MIME

Nieto Ramirez, N., & Nieto Londoño, R. D. (2019). *Implementación hardware de la función Hash SHA3-256 usando una arquitectura Pipeline.* Ingeniare. Obtenido de https://scielo.conicyt.cl/scielo.php?script=sci\_arttext&pid=S0718-33052019000100043

Pseudorandom\_Number\_Generator. (s.f.). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Pseudorandom\_number\_generator

Rainbow\_Table. (s.f.). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Rainbow\_table

Randomized\_Algorithms. (s.f.). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Randomized\_algorithm

Secure\_Hash\_Algorithms. (s.f.). Obtenido de Wikipedia: https://en.wikipedia.org/wiki/Secure\_Hash\_Algorithms