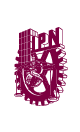
****

**INSTITUTO POLITÉCNICO NACIONAL**

**Unidad Profesional Interdisciplinaria de Ingeniería y Ciencias Sociales y Administrativas**

**Ingeniería en Informática**

*Seguridad Informática*

*Parcial 3*

HASH

Documento de Diseño

Estudiantes:

Estrada Pérez Hisao

Mecca Paredes Franco Samuel

Ríos Crisóstomo Omar

Grupos: 3 y 7

Profesor: Víctor Sandoval González

Secuencia: 3NM60

**UPIICSA - Octubre del 2019**

Contenido

[Introducción 2](#_Toc23970906)

[Algoritmo 3](#_Toc23970907)

[Propiedades 3](#_Toc23970908)

[Funcionamiento 4](#_Toc23970909)

[Codificación 4](#_Toc23970910)

[Paso 1: Descomponer la palabra 4](#_Toc23970911)

[Paso 2: Crear 4 bytes aleatorios 4](#_Toc23970912)

[Paso 3: Combinar bits de letras con bytes aleatorios y crear nuevos caracteres 4](#_Toc23970913)

[Paso 4: Bytes de configuración 5](#_Toc23970914)

[Decodificación 6](#_Toc23970915)

[Paso 1: Identificar la cantidad de datos codificada 6](#_Toc23970916)

[Paso 2: Decodificar cantidad de data 6](#_Toc23970917)

[Implementación 7](#_Toc23970918)

[Conclusión 8](#_Toc23970919)

# Introducción

El siguiente documento referencia una breve explicación del diseño del algoritmo HASH a desarrollar como parte del proyecto para el parcial 3 de la materia de Seguridad Informática.

# Algoritmo

La principal característica que se desea cumplir con el desarrollo del algoritmo es que este parezca lo mas aleatorio posible. Para lograrlo, se propondrá un esquema que cumpla con la característica de ser de un tamaño predeterminado, reduciendo cualquier rastro de alguna relación con el tamaño de la fuente.

Asimismo, se buscará que una palabra o dato se pueda representar de diferentes maneras, para reducir de esta manera las coincidencias con la fuente y hacer difícil su identificación.

A lo largo del diseño del algoritmo nos dimos cuenta que con las dos características principales a cumplir, el algoritmo parece ser robusto. Lamentablemente, por el tiempo de desarrollo con el que contamos, no se podrá implementar un algoritmo con el manejo de colisiones, por lo que el espectro de datos se reduce considerablemente.

Se logró identificar que el algoritmo cuenta con una generación de data aleatoria del 75%, que se explicará más adelante.

## Propiedades

Las principales propiedades que cumple el algoritmo son:

* Tamaño: 256 bytes (240 usables y 16 de configuración)
* Generación: aleatoria (75%)

## Funcionamiento

La explicación del funcionamiento del algoritmo se dará con un el siguiente ejemplo:

Codificar la palabra “Seguridad”

### Codificación

#### Paso 1: Descomponer la palabra

El primer paso descompone la palabra para trabajar cada byte a nivel de bits. De esta manera, en el caso de la letra S tendríamos los bits: 0101 0011

#### Paso 2: Crear 4 bytes aleatorios

Para garantizar la aleatoriedad se generan 4 bytes compuestos aleatoriamente excepto en los bits 32 y 4, esto debido a que las 32 primeras representaciones de caracteres tienen colisión con algunas instrucciones a nivel de procesador y el 4 para reducir el espectro de errores.

#### Paso 3: Combinar bits de letras con bytes aleatorios y crear nuevos caracteres

Se utilizan los 4 pares de bits de la enésima letra (en secuencia) para combinarlos con los 4 bytes generados aleatoriamente respectivamente de la siguiente manera:

El primer par de bits de la letra, en este caso el par 01 de los bits 128 y 64, se combina con el primer byte generado aleatoriamente, insertándole al byte creado los bits seleccionados de la letra en las posiciones 128 y 8. Teniendo como ejemplo generado el byte 0###1###, donde # representa 0 o 1.

De esta manera, por cada byte de la palabra se generarán 4 bytes para la cadena codificada.

#### Paso 4: Bytes de configuración

Debido a no manejar colisiones, nos vemos en el limite de manejar un tamaño de data limitado, por lo que se entregarán 240 bytes de data y 16 de configuración.

De la misma manera, se había ideado configurar la data para tomar los bits 128 y 8 (para un identificador de 0) y los bits 16 y 1 (para un identificador de 1) que ocuparía los bytes 129, 130, 131 y 132; e identificar mediante los 4 bytes siguientes (133 – 136) la cantidad de operaciones XOR, OR, AND y NOT, con un token que estaría constituido de las 4 primeras letras del identificador de usuario.

Pero debido a la complejidad y el poco tiempo disponible solo se tiene configurado en los bytes 121 – 128 la cantidad de data a codificar o decodificar (60 bytes).

Estos bytes de configuración los situamos al medio de toda la cadena, y de la misma manera serán codificados.

### Decodificación

#### Paso 1: Identificar la cantidad de datos codificada

El primer paso es tomar los bytes 121 – 128, decodificarlos e identificar la cantidad de data codificada.

#### Paso 2: Decodificar cantidad de data

Con la identificación de la cantidad de datos codificada, se toma el resto de la cadena y se decodifica, de manera inversa a la codificación, la cantidad exacta de datos tomando en cuenta la equivalencia de 1 byte sin codificar equivale a 4 bytes codificados.

# Implementación

En la carpeta implementación, se encuentra la implementación del diseño en el lenguaje de programación Python, y de la misma manera un archivo para poder utilizar los archivos como un servicio web.

# Conclusión

A pesar de la limitante de tiempo del proyecto, el algoritmo cumple con las expectativas mínimas de seguridad.