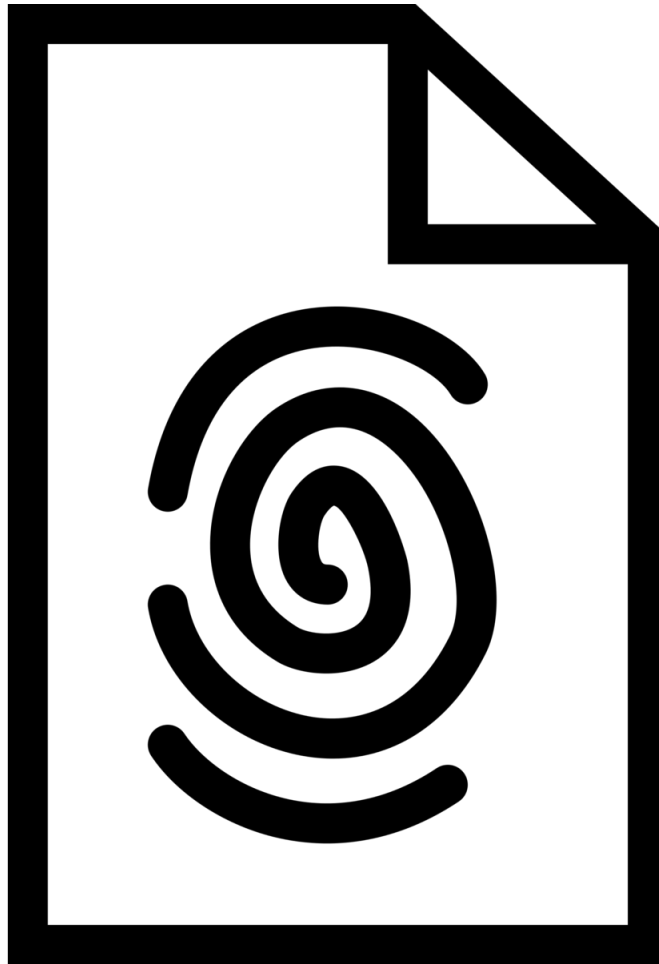


# UT3-T0 VERIFICACIÓN DE LA INTEGRIDAD DE ARCHIVOS MEDIANTE FUNCIONES HASH



Zakaria Chiouloud Boukhbiza

2º Administración de Sistemas Informáticos en Redes

15/11/2024



## ÍNDICE

<b>Introducción:</b> .....	<b>3</b>
<b>Diferencias entre MD5 y SHA-256:</b> .....	<b>4</b>
<b>Práctica en WINDOWS:</b> .....	<b>5</b>
Uso de QuickHash GUI : .....	9

# INTRODUCCIÓN:

En este informe, abordó el proceso de verificación de la integridad de archivos mediante funciones **hash** en los sistemas operativos **Windows** y **Linux**, explorando las diferencias entre ambas plataformas y las herramientas disponibles para generar hashes. Utilizaré **CertUtil** en **Windows**, una utilidad de línea de comandos que permite calcular el **hash** de archivos usando distintos algoritmos, como **MD5** y **SHA-256**. En **Linux**, emplearé los comandos **md5sum** y **sha256sum**, que son herramientas nativas y específicas para generar estos mismos **hashes**. A pesar de que el objetivo en ambas plataformas es el mismo, existen diferencias importantes en cuanto a la interfaz y el modo de ejecución de los comandos, lo cual puede afectar la experiencia del usuario en cada sistema operativo.

Las funciones **hash**, como **MD5** y **SHA-256**, generan una “**huella digital**” única de un archivo, que permite comprobar su integridad. **MD5** es un algoritmo más antiguo y rápido, pero menos seguro, mientras que **SHA-256** es más confiable en términos de seguridad, aunque requiere más recursos. Estas diferencias hacen que cada algoritmo sea útil en situaciones específicas: **MD5** puede ser conveniente para verificaciones rápidas en archivos de bajo riesgo, mientras que **SHA-256** es más adecuado para entornos en los que la seguridad es crítica.

Además de las herramientas de línea de comandos, existen aplicaciones de interfaz gráfica que facilitan el uso de estas funciones **hash**. En **Windows**, herramientas como **HashTab** o **QuickHash GUI** ofrecen una manera visual de generar **hashes**, lo que simplifica el proceso para los usuarios que prefieren evitar la terminal. Estas aplicaciones también están disponibles en **Linux**, ampliando la accesibilidad de las funciones **hash** para todos los usuarios. A través de esta práctica, demostraré cómo utilizar estas herramientas para confirmar la autenticidad de los archivos y reflejaré la importancia de los **hashes** en la seguridad informática actual, donde la protección de la integridad de los datos es fundamental.

# DIFERENCIAS ENTRE MD5 Y SHA-256:

## Diferencias entre MD5 y SHA-256:

### 1. Longitud del Hash:

- **MD5** genera un hash de 128 bits (32 caracteres en hexadecimal).
- **SHA-256** produce un hash de 256 bits (64 caracteres en hexadecimal), lo que significa una huella digital más extensa y, por tanto, más robusta frente a colisiones.

### 2. Seguridad y Vulnerabilidades:

- **MD5** es más vulnerable, ya que ha sido objeto de ataques de colisión, en los que dos entradas diferentes generan el mismo hash. Esto hace que no sea adecuado para aplicaciones donde la seguridad es crítica, como las firmas digitales.
- **SHA-256** es actualmente uno de los algoritmos hash más seguros para aplicaciones críticas, ya que no se conocen ataques prácticos que puedan generar colisiones en un tiempo razonable.

### 3. Velocidad y Eficiencia:

- **MD5** es más rápido que SHA-256, lo cual puede ser ventajoso en situaciones donde la velocidad es prioritaria y el nivel de seguridad no es crucial (como en la verificación rápida de integridad de archivos).
- **SHA-256**, debido a su diseño más complejo, es más lento en comparación, pero esta desaceleración está compensada por su mayor seguridad.

### 4. Aplicaciones Típicas:

- **MD5** se utiliza aún en situaciones no críticas, como la verificación de integridad de archivos en descargas, ya que es rápido y puede ayudar a identificar archivos alterados.
- **SHA-256** es comúnmente utilizado en entornos donde la seguridad es fundamental, como en criptografía, blockchain, y sistemas de autenticación seguros.

Estas diferencias hacen que **MD5** sea más adecuado para tareas de bajo riesgo y velocidad, mientras que **SHA-256** es la opción preferida para garantizar la integridad y seguridad en aplicaciones críticas.

# PRÁCTICA EN WINDOWS:

Utilicé la herramienta de línea de comandos **CertUtil** en Windows para calcular los hashes de un archivo de ejemplo (en formato .txt). Empleé los algoritmos MD5 y SHA-256 para demostrar la creación de huellas digitales y comparar los resultados de ambos métodos.

Estos serían los códigos HASH que da el archivo TXT que se llama "PruebaMD5":

**CertUtil -hashfile .\PruebaMD5.txt MD5**

**b306fadd881415b6ad2d4e739dad9299**

**CertUtil -hashfile .\PruebaMD5.txt SHA256**

**053ac0950d047c1948c2bea35340cc2ec55ac610def03a8f3d3305a3adee26c0**

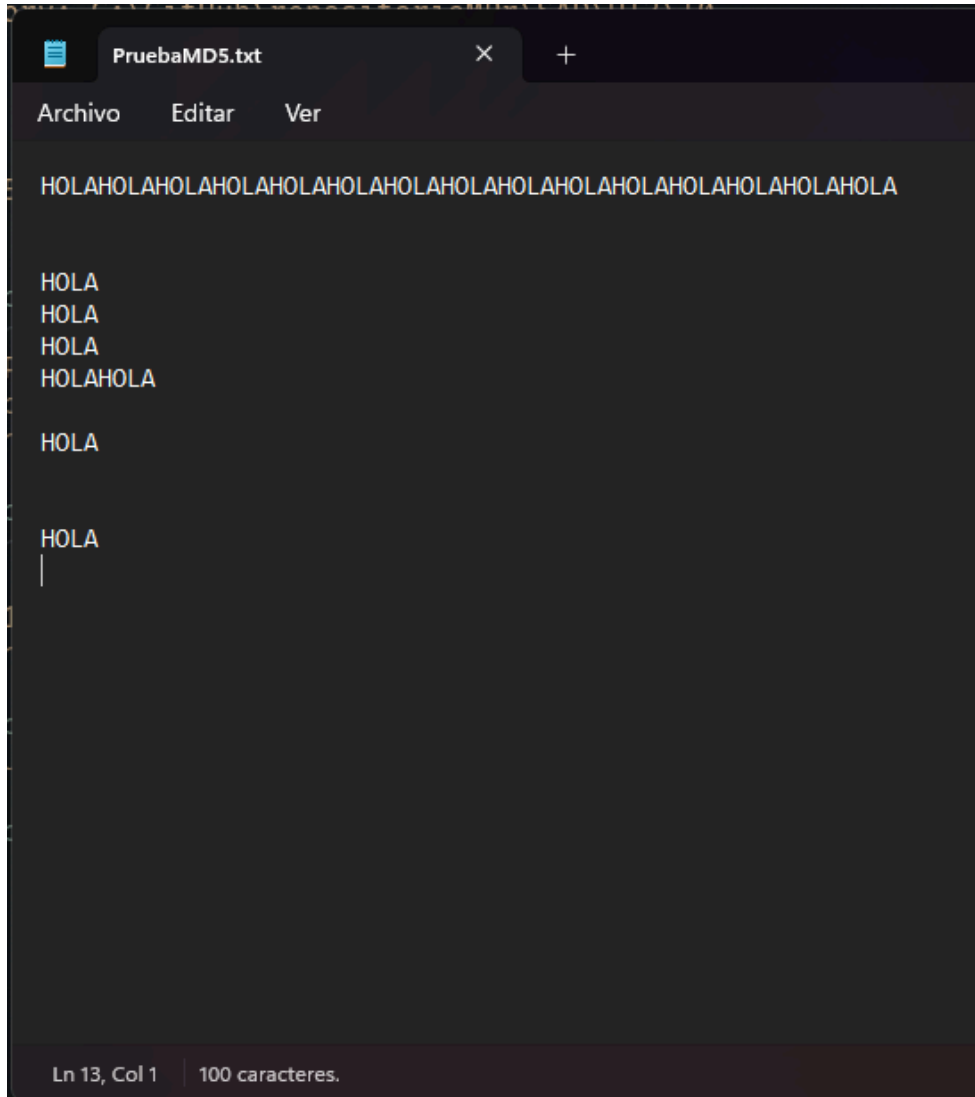
```
PowerShell x + v
[[@ zakar from zakaria][0.004s][RAM: 12/31GB][Friday at 5:01:26 AM][main = ?1]
[C:\GitHub\repositorioMPr\SAD\UT3\T0]
└─┬ LS

Directory: C:\GitHub\repositorioMPr\SAD\UT3\T0

Mode                LastWriteTime         Length Name
--                -
-a                15/11/2024   5:00             112 PruebaMD5.txt

[[@ zakar from zakaria][0.029s][RAM: 12/31GB][Friday at 5:01:27 AM][main = ?1]
[C:\GitHub\repositorioMPr\SAD\UT3\T0]
└─┬ CertUtil -hashfile .\PruebaMD5.txt MD5
MD5 hash de .\PruebaMD5.txt:
b306fadd881415b6ad2d4e739dad9299
CertUtil: -hashfile comando completado correctamente.
[[@ zakar from zakaria][0.171s][RAM: 12/31GB][Friday at 5:02:01 AM][main = ?1]
[C:\GitHub\repositorioMPr\SAD\UT3\T0]
└─┬ CertUtil -hashfile .\PruebaMD5.txt SHA256
SHA256 hash de .\PruebaMD5.txt:
053ac0950d047c1948c2bea35340cc2ec55ac610def03a8f3d3305a3adee26c0
CertUtil: -hashfile comando completado correctamente.
[[@ zakar from zakaria][0.09s][RAM: 12/31GB][Friday at 5:02:09 AM][main = ?1]
[C:\GitHub\repositorioMPr\SAD\UT3\T0]
└─┬
```

Este sería el archivo TXT al que hemos hecho el hash con MD5 y ahora SHA256

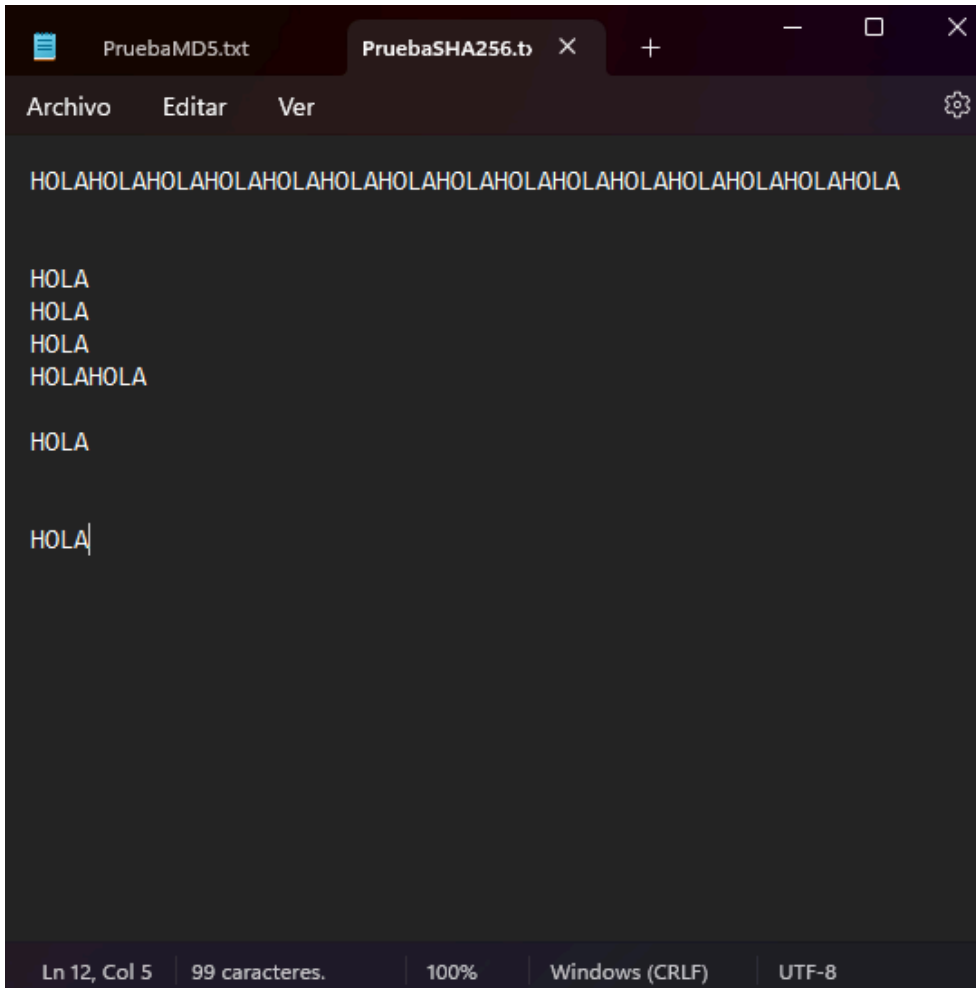


Trabajo: UT3- T0 - Verificación de la integridad de archivos mediante funciones hash

Asignatura: SAD

Zakaria Chiouloud Boukhbiza

Ahora vamos a hacer el Hash con el mismo documento pero modificando solo 1 carácter o espacio , cualquier mínimo cambio ya se hará un nuevo hash y vamos a verlo con este cambio que solo hemos quitado un espacio:



```
PruebaMD5.txt  PruebaSHA256.t  +  -  □  ×
Archivo  Editar  Ver  ⚙️
HOLAHOLAHOLAHOLAHOLAHOLAHOLAHOLAHOLAHOLAHOLAHOLAHOLAHOLAHOLAHOLAHOLA
HOLA
HOLA
HOLA
HOLAHOLA
HOLA
HOLA|
Ln 12, Col 5 | 99 caracteres. | 100% | Windows (CRLF) | UTF-8
```



Trabajo: UT3- T0 - Verificación de la integridad de archivos mediante funciones hash

Asignatura: SAD

Zakaria Chiouloud Boukhbiza

■ Aquí se ve con un mínimo cambio de un solo espacio ya hemos hecho que cree un nuevo HASH:

**CertUtil -hashfile .\PruebaSHA256.txt SHA256**

**c71ac2201e9fbc9230d771d36b45657d1396abf53b55beaf1c965e48678af84a**

**CertUtil -hashfile .\PruebaSHA256.txt MD5**

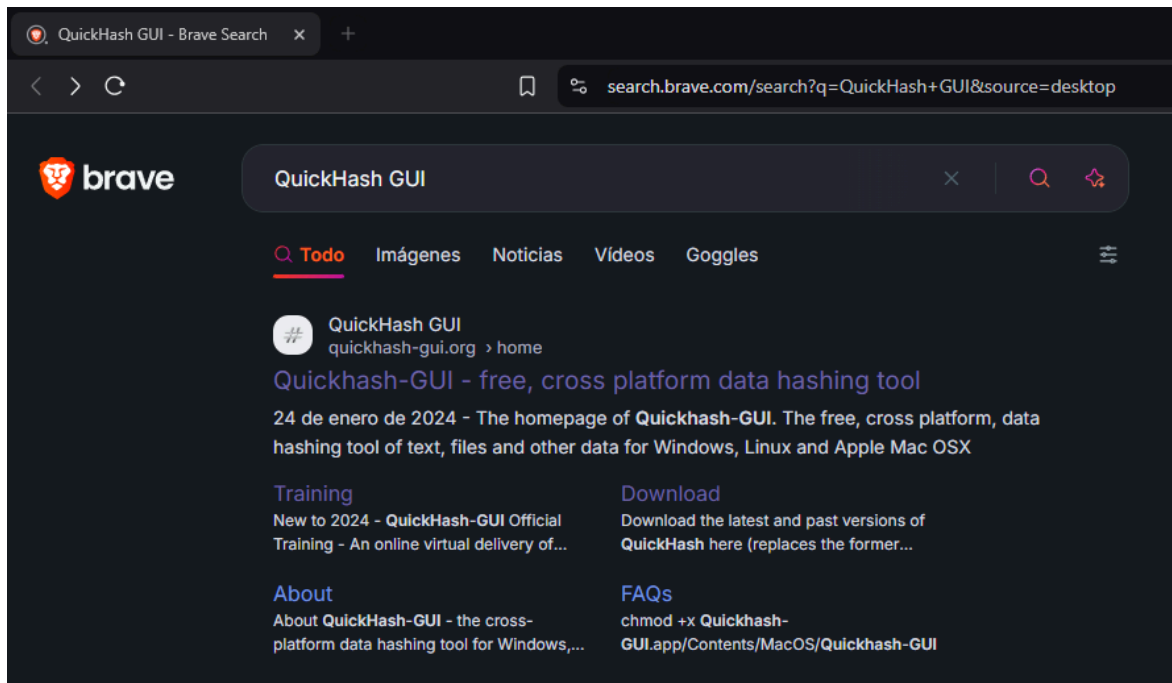
**7f8b6bd12b89cd37bb232f7e7ec08c7f**

```
└─┐ notepad PruebaMD5
[[@ zakar from ☐ zakaria][⌂ 0.14s][■ RAM: 12/31GB][📅 Friday at 5:08:36 AM][🔌 📶 main ≡ 🔌 ?1]
[C:\GitHub\repositorioMPr\SAD\UT3\T0]
└─┐ CertUtil -hashfile .\PruebaSHA256.txt SHA256
SHA256 hash de .\PruebaSHA256.txt:
c71ac2201e9fbc9230d771d36b45657d1396abf53b55beaf1c965e48678af84a
CertUtil: -hashfile comando completado correctamente.
[[@ zakar from ☐ zakaria][⌂ 0.139s][■ RAM: 13/31GB][📅 Friday at 5:21:24 AM][🔌 📶 main ≡ 🔌 ?1]
[C:\GitHub\repositorioMPr\SAD\UT3\T0]
└─┐ CertUtil -hashfile .\PruebaSHA256.txt MD5
MD5 hash de .\PruebaSHA256.txt:
7f8b6bd12b89cd37bb232f7e7ec08c7f
CertUtil: -hashfile comando completado correctamente.
[[@ zakar from ☐ zakaria][⌂ 0.1s][■ RAM: 13/31GB][📅 Friday at 5:21:40 AM][🔌 📶 main ≡ 🔌 ?1]
[C:\GitHub\repositorioMPr\SAD\UT3\T0]
└─┐
```

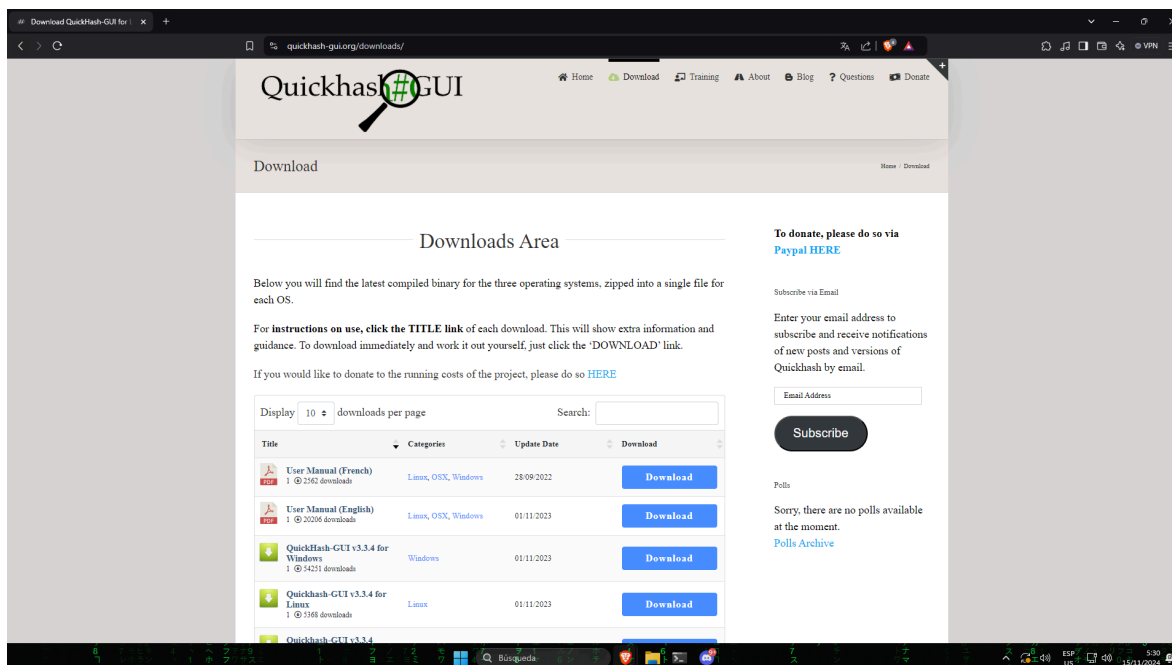


## USO DE QUICKHASH GUI :

Buscamos QuickHash GUI en el navegador y le damos a download:



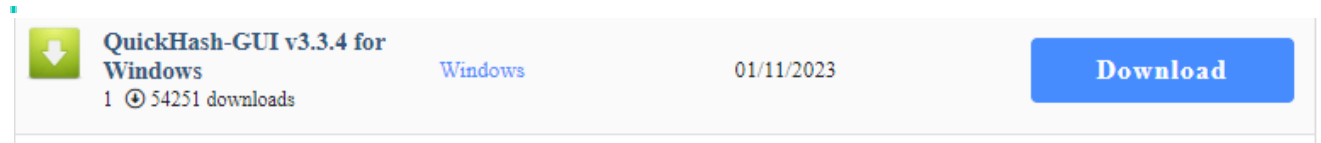
Y aqui elegimos el que pone que es para WINDOWS:



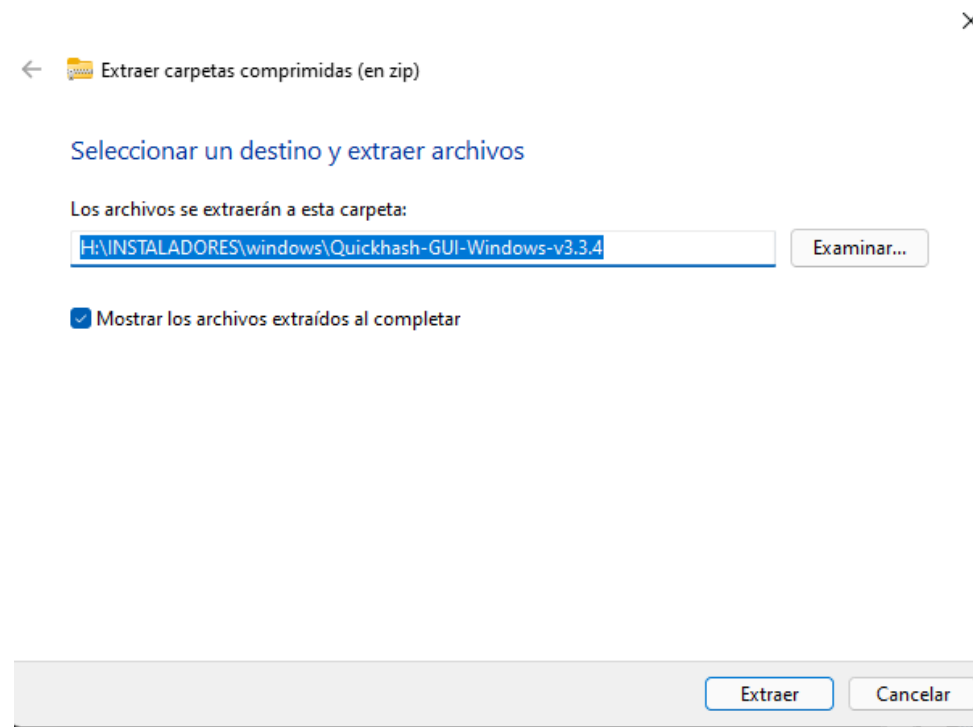
Trabajo: UT3- T0 - Verificación de la integridad de archivos mediante funciones hash

Asignatura: SAD

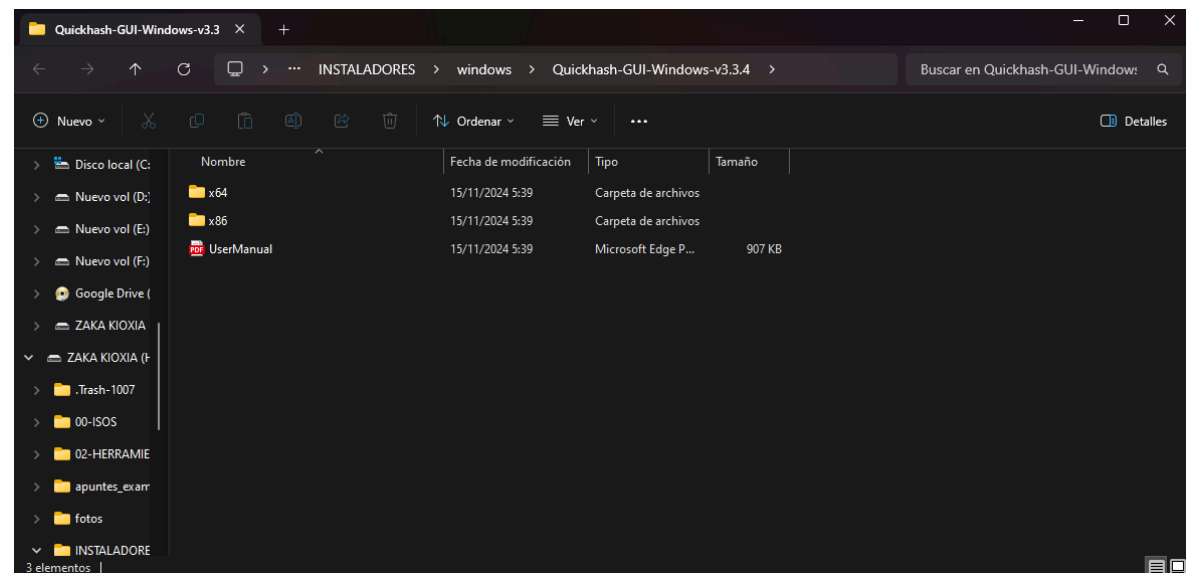
Zakaria Chiouloud Boukhbiza



Extraemos la carpeta comprimida:



Saldrá este contenido en la carpeta comprimida y entramos en x64:

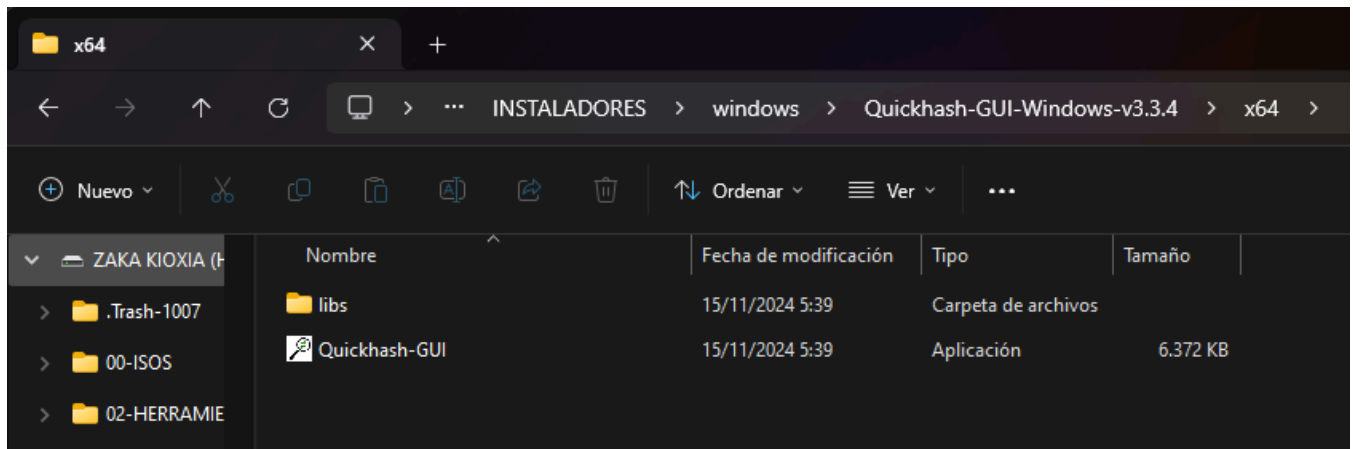


Trabajo: UT3- T0 - Verificación de la integridad de archivos mediante funciones hash

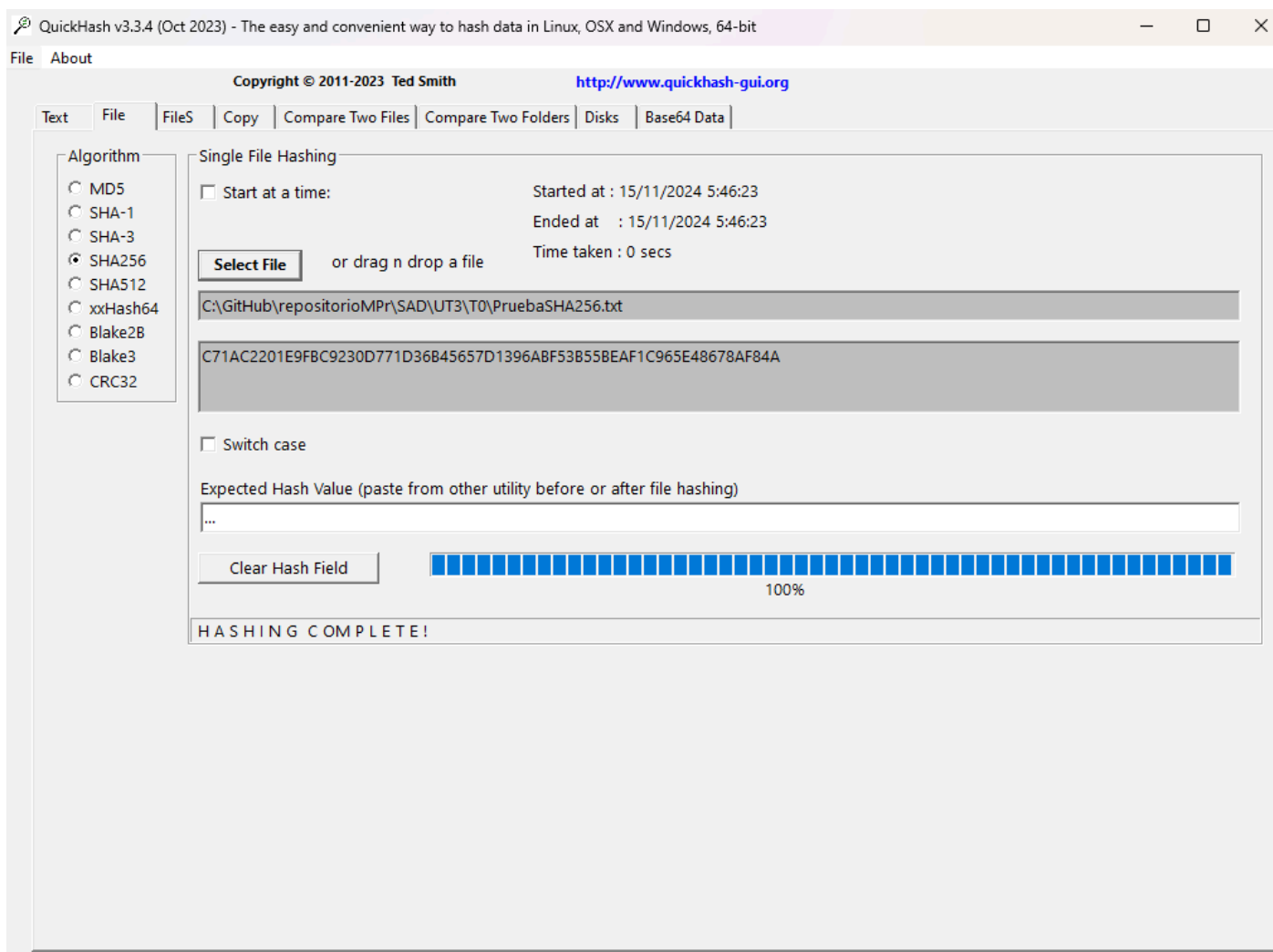
Asignatura: SAD

Zakaria Chiouloud Boukhbiza

Y aquí dentro abrimos la aplicación:



Cogemos la aplicación y ponemos el segundo archivo modificado y vemos que contenga el mismo código hash con la aplicación y lo verificaremos (Nos ha dado exactamente el mismo código HASH):

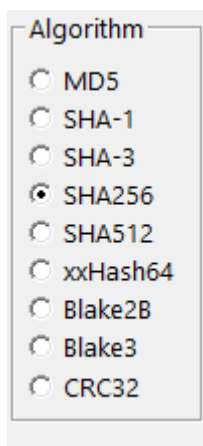


Trabajo: UT3- T0 - Verificación de la integridad de archivos mediante funciones hash

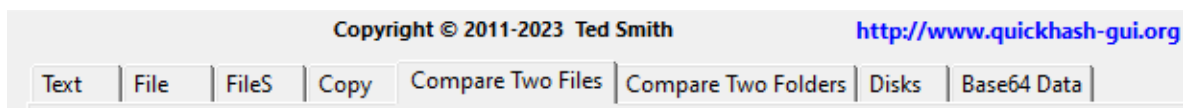
Asignatura: SAD

Zakaria Chiouloud Boukhbiza

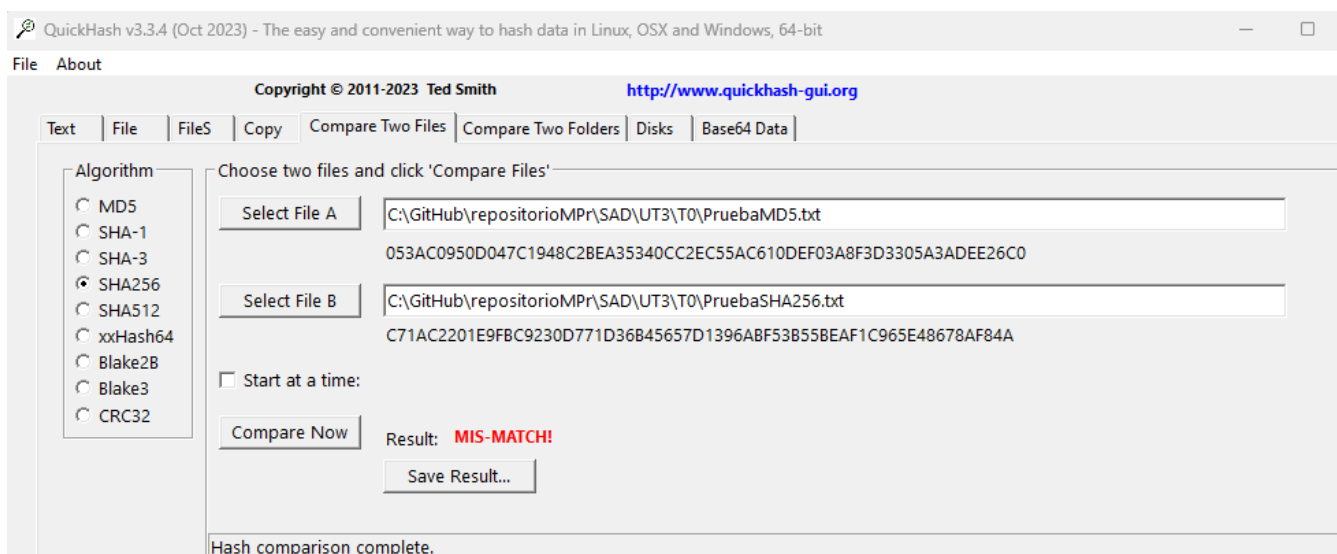
Aquí elegimos el Algoritmo que queremos utilizar:



Aquí tenemos las diferentes opciones que contiene el "Quickhash"



Aquí podremos ver si el código del archivo original ha sido modificado comparándolo con otro archivo "supuestamente igual" dándole a la opción de "compare two files" y veremos si ha sido o no modificado el contenido original:



# PRÁCTICA EN LINUX:

En linux (He usado la distribución Kali Blue) he usado **md5sum** y **sha256sum** como algoritmos para conseguir el código **hash** y creado un archivo primero **PruebaMD2sum.txt** y pusimos:

Usamos esta línea de comando para crear un archivo **“.txt”** y usamos primero **md5sum** y luego usaremos el algoritmo **hash sha256sum** :

```
$ echo "Aquí creo el nuevo documento y lo dejó creado" > PruebaMD2sum.txt
```

```
md5sum PruebaMD2sum.txt
```

y me da este **hash**

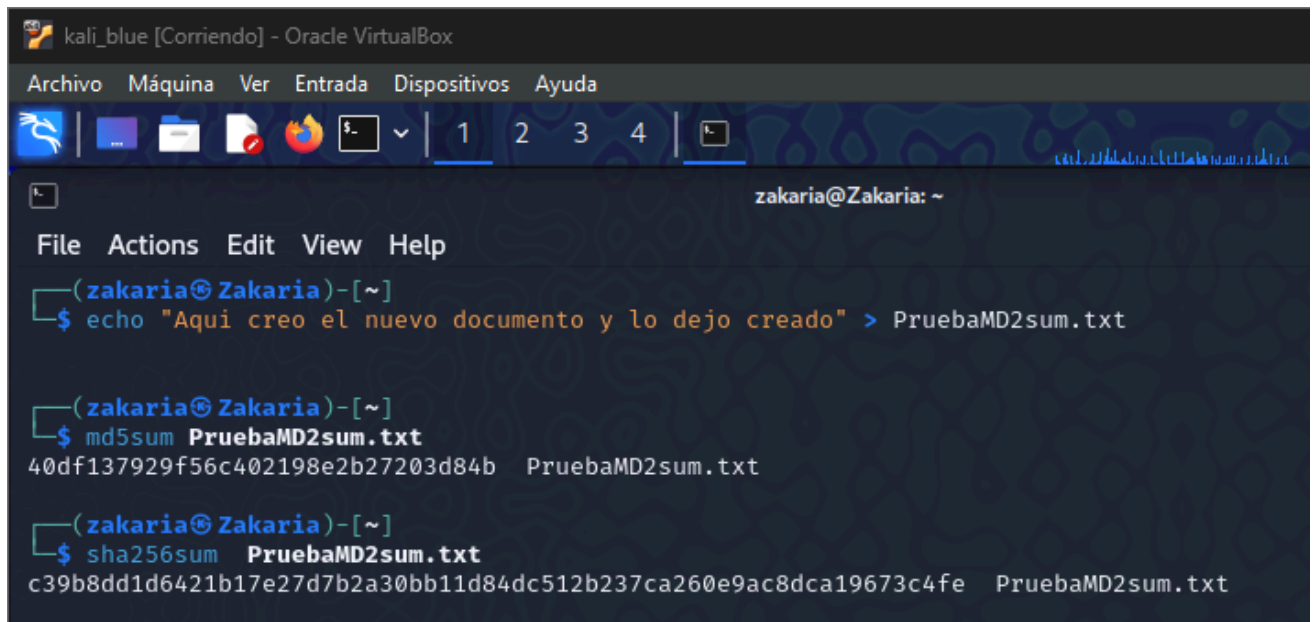
```
40df137929f56c402198e2b27203d84b PruebaMD2sum.txt
```

Y ahora usamos el algoritmo **sha256sum**:

```
sha256sum PruebaMD2sum.txt
```

```
c39b8dd1d6421b17e27d7b2a30bb11d84dc512b237ca260e9ac8dca19673c4fe
```

```
PruebaMD2sum.txt
```



```
kali_blue [Corriendo] - Oracle VirtualBox
Archivo  Máquina  Ver  Entrada  Dispositivos  Ayuda
[Icons] | [Icons] | [Icons] | [Icons] | [Icons] | 1 2 3 4 | [Icons]
zakaria@Zakaria: ~

File  Actions  Edit  View  Help

(zakaria@Zakaria)-[~]
$ echo "Aquí creo el nuevo documento y lo dejó creado" > PruebaMD2sum.txt

(zakaria@Zakaria)-[~]
$ md5sum PruebaMD2sum.txt
40df137929f56c402198e2b27203d84b PruebaMD2sum.txt

(zakaria@Zakaria)-[~]
$ sha256sum PruebaMD2sum.txt
c39b8dd1d6421b17e27d7b2a30bb11d84dc512b237ca260e9ac8dca19673c4fe PruebaMD2sum.txt
```

Trabajo: UT3- T0 - Verificación de la integridad de archivos mediante funciones hash

Asignatura: SAD

Zakaria Chiouloud Boukhbiza

Ahora copiamos el archivo y cambiamos el contenido con “**nano**” y cambiamos el archivo de dentro por esto:

**cp PruebaMD2sum.txt Pruebasha256sum.txt**

```
(zakaria@Zakaria)-[~]
$ cp PruebaMD2sum.txt Pruebasha256sum.txt
```

**sudo nano Pruebasha256sum.txt**

```
zakaria@Zakaria: ~
File Actions Edit View Help
GNU nano 8.2 Pruebasha256sum
Aquí creo el nuevo documento y lo dejo creado y modificado
```

Después de copiar el archivo anterior, cambiar el contenido y cambiarle el nombre para diferenciarlo podemos comprobar los hash:

Primero con el algoritmo **sha256sum**:

**sha256sum Pruebasha256sum.txt**

**7e0e471f69621e1ac2863489b52ed733acdbfa23cfc1b581aedc8589aba0966d**

**Pruebasha256sum.txt**

Y ahora con el algoritmo **md5sum**:

**md5sum Pruebasha256sum.txt**

**ef202f2d1f1f858b7ea3c86f94eb4e3e Pruebasha256sum.txt**

```
(zakaria@Zakaria)-[~]
$ cp PruebaMD2sum.txt Pruebasha256sum.txt

(zakaria@Zakaria)-[~]
$ sudo nano Pruebasha256sum.txt
[sudo] password for zakaria:

(zakaria@Zakaria)-[~]
$ sha256sum Pruebasha256sum.txt
7e0e471f69621e1ac2863489b52ed733acdbfa23cfc1b581aedc8589aba0966d Pruebasha256sum.txt

(zakaria@Zakaria)-[~]
$ md5sum Pruebasha256sum.txt
ef202f2d1f1f858b7ea3c86f94eb4e3e Pruebasha256sum.txt
```

# ANÁLISIS:

## ¿Por qué los algoritmos MD5 y SHA-1 ya no son recomendados para aplicaciones críticas?

Los algoritmos MD5 y SHA-1 se consideran inseguros debido a vulnerabilidades específicas que comprometen su integridad. Estos problemas se deben principalmente a su debilidad frente a **ataques de colisión**, donde dos entradas diferentes producen el mismo hash. Las principales razones son:

### Ataques de colisión en MD5 y SHA-1:

- **MD5:** En 2004, se demostró la posibilidad de generar colisiones de manera práctica, es decir, dos archivos diferentes con el mismo hash. Esto ha sido explotado para falsificar certificados digitales o documentos críticos.
- **SHA-1:** Aunque es más seguro que MD5, en 2017 Google y CWI Amsterdam realizaron un ataque práctico que generó una colisión, probando su vulnerabilidad.

### Fuerza computacional:

- Los avances en hardware (por ejemplo, GPUs potentes) han reducido significativamente el costo de realizar estos ataques, haciendo que estos algoritmos sean más fáciles de romper.

### Estándares modernos:

- Organizaciones como NIST (National Institute of Standards and Technology) han dejado de recomendar MD5 y SHA-1 para aplicaciones criptográficas críticas, prefiriendo algoritmos más seguros como **SHA-256** o **SHA-3**.

## ¿Cuándo sería aceptable utilizar MD5 en lugar de SHA-256 o SHA-512?

Aunque MD5 no es seguro para aplicaciones críticas, aún puede ser aceptable en ciertas situaciones donde la velocidad sea prioritaria y la seguridad no sea crítica:

### 1. Verificación de integridad básica:

- Para verificar que un archivo no se ha corrompido durante la transferencia, siempre que la fuente sea confiable y no exista un riesgo de ataque.
- Por ejemplo, al comprobar la integridad de grandes volúmenes de datos dentro de un sistema interno.

### 2. Hashing de datos no sensibles:

- En aplicaciones donde los datos no están relacionados con la seguridad, como crear identificadores únicos de archivos (ejemplo: generación de checksums en sistemas de almacenamiento).

### 3. Sistemas legados:

- En sistemas antiguos donde no es posible implementar algoritmos más modernos debido a limitaciones técnicas.

---

## REFLEXIÓN FINAL:

Las funciones **hash** son fundamentales para garantizar la integridad y autenticidad de los datos. Su uso correcto asegura que los archivos no han sido alterados, previniendo ataques y manipulaciones. A pesar de sus limitaciones, el conocimiento de algoritmos como **MD5**, **SHA-1** y **SHA-256** permite tomar decisiones informadas al seleccionar herramientas de seguridad adecuadas.