Ejercicio en clase de funciones de hash

Marco Jurado 20308

Investigación

Las funciones hash son aquellas que toman una entrada y van a devolver una cadena de longitud fija el cual es único dependiendo de cada uno de los caracteres de la entrada. Es por esto que dichas funciones nos devolveran un codigo único para cada entrada aún así le cambiemos lo más mínimo. Esto permite que podamos identificar cambios en cualquier bit del mismo y tomar las debidas precauciones para poder determinar si el archivo enviado o recibido corresponde al cual nos estan enviando. Este proceso se utiliza para poder realizar la verificación de cualquier tipo de documento o texto hasheado con la función para poder determinar la veracidad y la integridad de la información contenida en el mismo.

Algunas de las funciones hash más comunes son: MD5, SHA 1, SHA 2, SHA 3 y Blake. Estas han sido utilizadas y cuentan con sus pequeñas fortaleces y debilidades. Por ejemplo en el caso del algoritmo MD5 se cuenta con la vulnerabilidad de ataques de colisión. Esto es generado u ocasionado al saber que el algoritmo genera codigos hash iguales y esto crea una colisión por lo cual el mismo algoritmo no es seguro (Gewarren, 2023). Si hacemos una comparación entre las funciones hash MD5 y SHA-256 podemos mencionar lo siguiente:

- MD5: Produce un valor hash de 128 bits. Fue ampliamente utilizado debido a su velocidad y simplicidad. Sin embargo, ha sido vulnerado en términos de seguridad y ya no se considera seguro para aplicaciones críticas.
- SHA-256: Parte de la familia SHA-2, produce un valor hash de 256 bits, lo que le proporciona una mayor seguridad frente a ataques de colisión (donde dos entradas distintas producen el mismo valor hash).

Ahora vamos a realizra una comparación de tiempo de la función MD5 y SHA-256 utilizando el archivo 'documento.txt' adjunto al ejercicio.

In [ ]:
```
import hashlib
import time
```

In [ ]:
```
def read_file_as_string(file_path):
    try:
        with open(file_path, 'r', encoding='utf-8') as file:
            content = file.read()
        return content
    except FileNotFoundError:
        return "El archivo no fue encontrado."
```

```
    except Exception as e:
        return f"Error al leer el archivo: {e}"
```

In [ ]:
```
def hash_md5(input_string):
    md5_hash = hashlib.md5()
    input_bytes = input_string.encode('utf-8')
    md5_hash.update(input_bytes)
    hashed_string = md5_hash.hexdigest()
    return hashed_string

input_data = "Ejemplo de cadena a hashear"
hashed_data = hash_md5(input_data)
print("Datos originales:", input_data)
print("Hash MD5:", hashed_data)
```

```
Datos originales: Ejemplo de cadena a hashear
Hash MD5: f285a0187af860109ed1e685cd4ac4be
```

In [ ]:
```
def hash_sha256(input_string):
    sha256_hash = hashlib.sha256()
    input_bytes = input_string.encode('utf-8')
    sha256_hash.update(input_bytes)
    hashed_string = sha256_hash.hexdigest()
    return hashed_string
```

In [ ]:
```
documento_original_path = 'documento.txt'
file_content = read_file_as_string(documento_original_path)
```

In [ ]:
```
def compare_hashes(input_string):
    start_time_md5 = time.time()
    result_md5 = hash_md5(input_string)
    end_time_md5 = time.time()
    time_md5 = end_time_md5 - start_time_md5

    start_time_sha256 = time.time()
    result_sha256 = hash_sha256(input_string)
    end_time_sha256 = time.time()
    time_sha256 = end_time_sha256 - start_time_sha256

    return result_md5, time_md5, result_sha256, time_sha256

input_data = "Ejemplo de cadena a hashear para comparación de rendimiento"
md5_result, md5_time, sha256_result, sha256_time = compare_hashes(input_data)

print("MD5 Hash:", md5_result)
print("MD5 Time (seconds):", md5_time)
print("SHA-256 Hash:", sha256_result)
print("SHA-256 Time (seconds):", sha256_time)
```

```
MD5 Hash: cd5100aa810c4aef6568b252f6e90269
MD5 Time (seconds): 0.0
SHA-256 Hash: 779cd860b009963aff1df63645b72caf36aab997581210e0a7ac765f13a9b35e
SHA-256 Time (seconds): 0.0
```

Bibliografía

- Gomez, E. E. P. (2023, May 19). MD5 vs sha-1 vs SHA-2: ¿cuál es el hash cifrado más seguro y cómo verificarlos?. freeCodeCamp.org. https://www.freecodecamp.org/espanol/news/md5-vs-sha-1-vs-sha-2-cual-es-el-hash-cifrado-mas-seguro-y-como-verificarlos/

- Donohue, B., Stern, A., Grustniy, L., Svistunova, O., & Diazgranados, H. (n.d.). ¿Qué es un hash y cómo funciona?. Daily Spanish LatAm latamkasperskycomblog. https://latam.kaspersky.com/blog/que-es-un-hash-y-como-funciona/2806/

- Gewarren. (n.d.). CA5351: No Usar Algoritmos Criptográficos Dañados (Análisis de Código) - .NET. CA5351: No usar algoritmos criptográficos dañados (análisis de código) - .NET | Microsoft Learn. https://learn.microsoft.com/es-es/dotnet/fundamentals/code-analysis/quality-rules/ca5351

Aplicación

In [ ]:
```
file_content = read_file_as_string('documento.txt')
hash_original = hash_sha256(file_content)
hash_original
```

Out[ ]:
```
'29f68526129c28be4a3b01d5a4ff5e5e92ff5a05b697f736fa08101e7cdcd1ad'
```

In [ ]:
```
file_content = read_file_as_string('documento_1.txt')
hash_documento1 = hash_sha256(file_content)
hash_documento1
```

Out[ ]:
```
'679edb4a383ff223070004635095f0c12610171627280c646df70c068a8a0d50'
```

In [ ]:
```
file_content = read_file_as_string('documento_2.txt')
hash_documento2 = hash_sha256(file_content)
hash_documento2
```

Out[ ]:
```
'1b99376c85bad1b005e83cd03d6af5302b6a53769b806f9d5f6d57abb99d4df7'
```

Corroboración

In [ ]:
```
with open('hash_original.txt', 'r', encoding='utf-8') as file:
    hash_verdadero = file.read()

hash_verdadero
```

Out[ ]:
```
'29f68526129c28be4a3b01d5a4ff5e5e92ff5a05b697f736fa08101e7cdcd1ad'
```

In [ ]:
```python
def compare_hashes_with_original(hash_original, hashes_paraComprobar):
    results = {}
    for file_path, generated_hash in hashes_paraComprobar.items():
        results[file_path] = (generated_hash == hash_original)
    return results
```

In [ ]:
```python
hashes_paraComprobar = {
    'documento.txt': hash_original,
    'documento_1.txt': hash_documento1,
    'documento_2.txt': hash_documento2
}

resultados_comparacion_hashes = compare_hashes_with_original(hash_verdadero, hashes
for file, result in resultados_comparacion_hashes.items():
    if result:
        print(f"Archivo: {file}, Resultado: Coincide con el hash original.")
    else:
        print(f"Archivo: {file}, Resultado: NO coincide con el hash original.")
```

```
Archivo: documento.txt, Resultado: Coincide con el hash original.
Archivo: documento_1.txt, Resultado: NO coincide con el hash original.
Archivo: documento_2.txt, Resultado: NO coincide con el hash original.

Archivo: documento_1.txt, Resultado: NO coincide con el hash original.
Archivo: documento_2.txt, Resultado: NO coincide con el hash original.
```

Ahora que sabemos que los archivos documento_1.txt y documento_2.txt NO coinciden con el hash original sabemos que estos son archivos que fueron corruptos en su contenido y no deberían de ser aceptados como validos. Por lo tanto vamos a observar que diferencias existen en ellos.

documento_1.txt

Como podemos ver en el documento de texto de documento_1 Sabemos que es muy obvio que hay una gran diferencia en el contenido del texto. En este caso Hacen falta 3 parrafos de contenido explicando así la diferencia en el hash obtenido. Esto es interesante pues a pesar de tener menor cantidad de bits de contenido en el archivo la función nos devuelve siempre un hash de longitud definida y estandar.

documento_2.txt

En este caso la diferencia es la más mínima. En esta la única diferencia determinada es la ausencia de una tilde en la palabra Minería. Esto quiere decir que no importa el más minimo cambio en el documento en el contenido del mismo, siempre tendremos un hash distinto. Esto hace meramente seguro poder determinar y validar si un archivo es el esperado siempre.