Lab Report: Hashing in Cryptography

Date: 2025 | 04 | 08
Name: S.N.M Gedara [29165]

Task 1: Generating Hash Values

Hash a Simple String

```
import hashlib

def generate_hash(message, algorithm='sha256'):
    hash_function = hashlib.new(algorithm)
    hash_function.update(message.encode('utf-8'))
    return hash_function.hexdigest()

message = "Hello, NSBM!"
print("SHA-256 Hash:", generate_hash(message, 'sha256'))
print("MD5 Hash:", generate_hash(message, 'md5'))
```

Observations:

- **SHA-256 Hash:**[ab3af23b5f777ed7af403aee0447a2b01ede1ae875896f12741a1e3175e96731]
- **MD5 Hash:** [bbe48d90ad52a69eef332c48d41b93ec]

Modify Message to ‘Hello,NSBM?”

```python
import hashlib

Tabnine | Edit | Test | Explain | Document
def generate_hash(message, algorithm='sha256'):
    hash_function = hashlib.new(algorithm)
    hash_function.update(message.encode('utf-8'))
    return hash_function.hexdigest()

message = "Hello, NSBM?"
print("SHA-256 Hash:", generate_hash(message, 'sha256'))
print("MD5 Hash:", generate_hash(message, 'md5'))
```

```
[Running] python -u "d:\3 year 1 sem\CRYPTO\LAB\Hashing\generatingvalues.py"
SHA-256 Hash: f499c86668304dbec4f886d89a7880c0a5d7e52274762183a7cd2898fc6e1870
MD5 Hash: 555ca6e9a4c4e83897887e38c87957dc
```

Observations:

- **SHA-256 Hash:**[f499c86668304dbec4f886d89a7880c0a5d7e52274762183a7cd2898fc6e1870]
- **MD5 Hash:** [555ca6e9a4c4e83897887e38c87957dc]

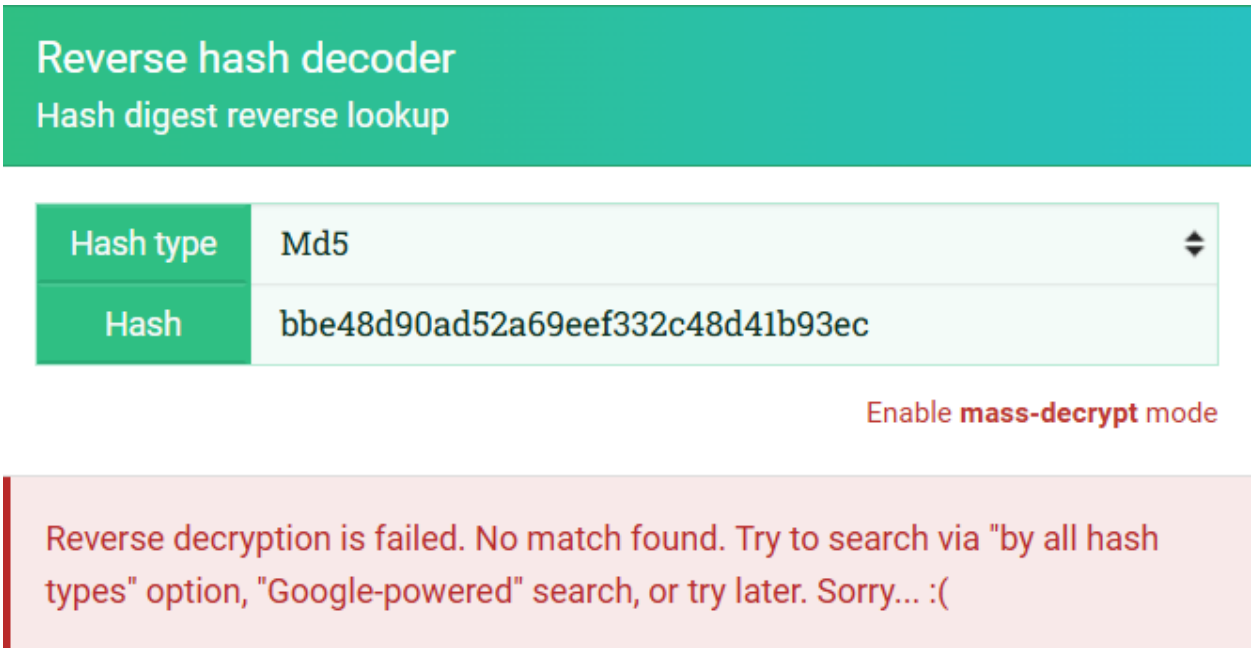
When we hash a string, even little changes to the input message result in an entirely different hash value. This is an important characteristic of cryptographic hash functions: slight changes in the input should result in significantly distinct hash outputs.

```
Hello, NSBM!
SHA-256 Hash: ab3af23b5f777ed7af403aee0447a2b01ede1ae875896f12741a1e3175e96731
MD5 Hash: bbe48d90ad52a69eef332c48d41b93ec
----------------------------------------------------------------
Hello, NSBM?
SHA-256 Hash: f499c86668304dbec4f886d89a7880c0a5d7e52274762183a7cd2898fc6e1870
MD5 Hash: 555ca6e9a4c4e83897887e38c87957dc
```

Task 2: Pre-Image Resistance Test

Using: https://md5hashing.net/hash Web site, Tried to reverse-engineer the hash :

MD5 Hash: bbe48d90ad52a69eef332c48d41b93ec



Question: Can you retrieve the original input from the hash? Why or why not?

We cannot get the original input from the hash. A cryptographic hash function is a one-way function, which means that once data is changed into a hash, it cannot be returned to its original form.

Task 3: Second Pre-Image Resistance Test

```
Hello, NSBM!
SHA-256 Hash: ab3af23b5f777ed7af403aee0447a2b01ede1ae875896f12741a1e3175e96731
MD5 Hash: bbe48d90ad52a69eef332c48d41b93ec
----------------------------------------------------------------------------------------
Hello, NSBM?
SHA-256 Hash: f499c86668304dbec4f886d89a7880c0a5d7e52274762183a7cd2898fc6e1870
MD5 Hash: 555ca6e9a4c4e83897887e38c87957dc
----------------------------------------------------------------------------------------
Hello, NSBM$
SHA-256 Hash: 3d664d02ae540f9d757ec5a833fa0fa8e6c4395be57289faf83575790cfc09c4
MD5 Hash: a63640583aa94dd06db8fecdb00a35b2
----------------------------------------------------------------------------------------
Hello, NSBM*
SHA-256 Hash: e41d817e3c235c02615c55903c9ecc3b524d2930296661c7930a941a95a77529
MD5 Hash: a4eb120660d160ff7fb15215a7a9ada8
```

Question: Were you able to find another input with the same hash? Why is this important for security?

We cannot find another input with the same hash. A secure hash function should prevent two separate inputs from generating the same hash (second pre-image resistance).

Task 4: Collision Resistance Test

```
def generate_hash(message, algorithm='sha256'):
    hash_function = hashlib.new(algorithm)
    hash_function.update(message.encode('utf-8'))
    return hash_function.hexdigest()

message = "Cryptography"
print("Cryptography")
print("SHA-256 Hash:", generate_hash(message, 'sha256'))
print("MD5 Hash:", generate_hash(message, 'md5'))
print("----------------------------------------------------------------------------------------")
message = "CRYPTOGRAPHY"
print("CRYPTOGRAPHY")
print("SHA-256 Hash:", generate_hash(message, 'sha256'))
print("MD5 Hash:", generate_hash(message, 'md5'))
print("----------------------------------------------------------------------------------------")
message = "CRYPTOGRAPHY"
print("Modify the message")
print("CRYPTOGRAPHY!")
print("SHA-256 Hash:", generate_hash(message, 'sha256'))
print("MD5 Hash:", generate_hash(message, 'md5'))
```

```
Cryptography
SHA-256 Hash: b584eec728548aced5a66c0267dd520a00871b5e7b735b2d8202f86719f61857
MD5 Hash: 64ef07ce3e4b420c334227eecb3b3f4c
----------------------------------------------------------------------------------------
CRYPTOGRAPHY
SHA-256 Hash: b8925bb454caeaa4dc632abffa9cfec14e42c037639d804bb9ad22eea425e935
MD5 Hash: 77e3034cb76af169150cf14dd657cb19
----------------------------------------------------------------------------------------
Modify the message
CRYPTOGRAPHY!
SHA-256 Hash: b8925bb454caeaa4dc632abffa9cfec14e42c037639d804bb9ad22eea425e935
MD5 Hash: 77e3034cb76af169150cf14dd657cb19
```

Question: Did any two different messages produce the same hash?

Even minor modifications to the message, such as using capital instead of lowercase, result in an entirely different hash. This illustrates collision resistance, with each unique input producing a distinct hash value.

Research:

MD5 collision attack example:

In 2017, researchers revealed that MD5 collisions may be manufactured in less than a minute, indicating that MD5 is vulnerable to collision assaults. This was part of Google's SHAttered assault. *Stevens, Marc. (2006).*

Task 5: HMAC (Hash-Based Message Authentication Code)

Code Used:

```
import hmac
import hashlib

def generate_hmac(key, message, algorithm='sha256'):
    return hmac.new(key.encode(), message.encode(), hashlib.new(algorithm).name).hexdigest()

key = "securekey123"
message = "Authenticate this message"

print("HMAC (SHA-256):", generate_hmac(key, message, 'sha256'))
```

HMAC Output:

- **HMAC (SHA-256):**
  HMAC output for "Authenticate this message" and key "securekey123"

  [94991541e2fb4158965b5821072a97b50dae10ec442ce6e65b81e43bde989040]

Modify the key:

New key: “nsbm45” | message: "Authenticate this message”

HMAC(SHA-256) :
[8c861716230474dbdf113733f5b9bddfba2bd7da18f7951b9b58b33d9a089eb1]

Modify the Message:

New message: “Welcome to NSBM” | key = "securekey123"

HMAC(SHA-256) :

[2b94cd23d2f94aa63bfeebc1028d35c1f2fd99fbc4fadff46cdc3dcee1ee0706]

```
Modify Key
HMAC (SHA-256): 8c861716230474dbdf113733f5b9bddfba2bd7da18f7951b9b58b33d9a089eb1
--------------------------------------------------------------------------
Modify the Message
HMAC (SHA-256): 2b94cd23d2f94aa63bfeebc1028d35c1f2fd99fbc4fadff46cdc3dcee1ee0706
```

Changing the key or message changes the HMAC value, demonstrating how HMAC protects both message integrity and authentication.

HMAC uses both a secret key and a hash function, ensuring that the message is both authentic and untampered.

Key findings:

Pre-image resistance prevents retrieval of the original input from the hash.
Second, pre-image resistance precludes discovering an alternative message that generates the same hash.
Collision resistance guarantees that two separate messages do not produce the identical hash.
HMAC adds security by combining a secret key with hashing to ensure message integrity and authenticity.

Questions:

1. What are the three main security properties of a cryptographic hash function?
2. How does collision resistance protect against fraudulent file replacements?
3. Why is SHA-256 preferred over MD5?
4. Explain how HMAC differs from normal hashing.
5. Research and describe a real-world attack where a weak hash function was exploited.

Answer:

1]

- Pre-image resistance
- Second pre-image resistance
- Collision resistance

2]

Collision resistance guarantees that no two separate files generate the same hash. If an attacker attempts to replace a file, the hash will change, making fraudulent alterations easier to detect.

3]

SHA-256 is thought to be more secure than MD5, which contains weaknesses that allow for collision attacks. These vulnerabilities do not affect SHA-256.

4]

HMAC generates a hash using a secret key as well as the input message. This makes HMAC more secure for authentication since the message can only be verified by the recipient who has the secret key.

5]

One example is Google's SHAttered attack, which used an MD5 collision to fabricate a digital certificate. This attack demonstrated that MD5 is not collision resistant and should not be utilised for cryptographic security.

[*reference :* *https://www.techtarget.com/searchsecurity/tip/SHA-1-collision-How-the-attack-completely-breaks-the-hash-function*]