Python Hashing Tutorial

Contents

Time required: 60 minutes

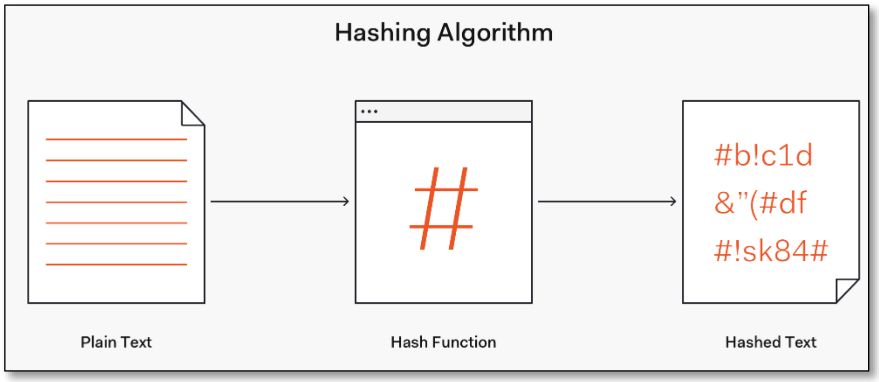
What is Hashing?

Hashing is a process of converting input data (also known as plaintext) into a fixed-size string of characters, typically a sequence of letters and numbers, using a mathematical algorithm called a hash function. This output string is often referred to as a hash or message digest.

- **One-Way Function**: A hash function is designed to be a one-way function, meaning it's easy to compute the hash value from the input (plaintext), but extremely difficult (ideally impossible) to reverse-engineer the original input from the hash.
- **Fixed Output Size**: Regardless of the input size, a hash function produces an output of fixed length. For instance, the SHA-256 algorithm always generates a 256-bit hash value, regardless of the length of the input.
- **Deterministic**: For the same input, a hash function will always produce the same hash value. Even a minor change in the input will result in a substantially different hash output.
- **Collision Resistance**: A good hash function minimizes the chances of two different inputs producing the same hash output. This property is called collision resistance and is a crucial aspect of secure hashing algorithms.

Uses of Hashing in Cybersecurity

- **Password Storage**: In cybersecurity, hashing is commonly used to store passwords securely. Instead of storing the actual passwords, systems store the hash of the passwords. During authentication, the entered password's hash is compared with the stored hash for verification.
- **Data Integrity**: Hashes are used to verify the integrity of data. By generating hashes of files or messages, one can verify if the data has been altered during transmission or storage.
- **Digital Signatures**: Hashing is a fundamental component of digital signatures. It helps ensure the authenticity and integrity of digital documents by creating a unique hash of the document and then encrypting that hash using a private key.
- **Forensic Analysis**: Ethical hackers may use hashing to verify the integrity of files, analyze malware, or identify known malicious files by comparing their hashes with a database of known malicious hashes.

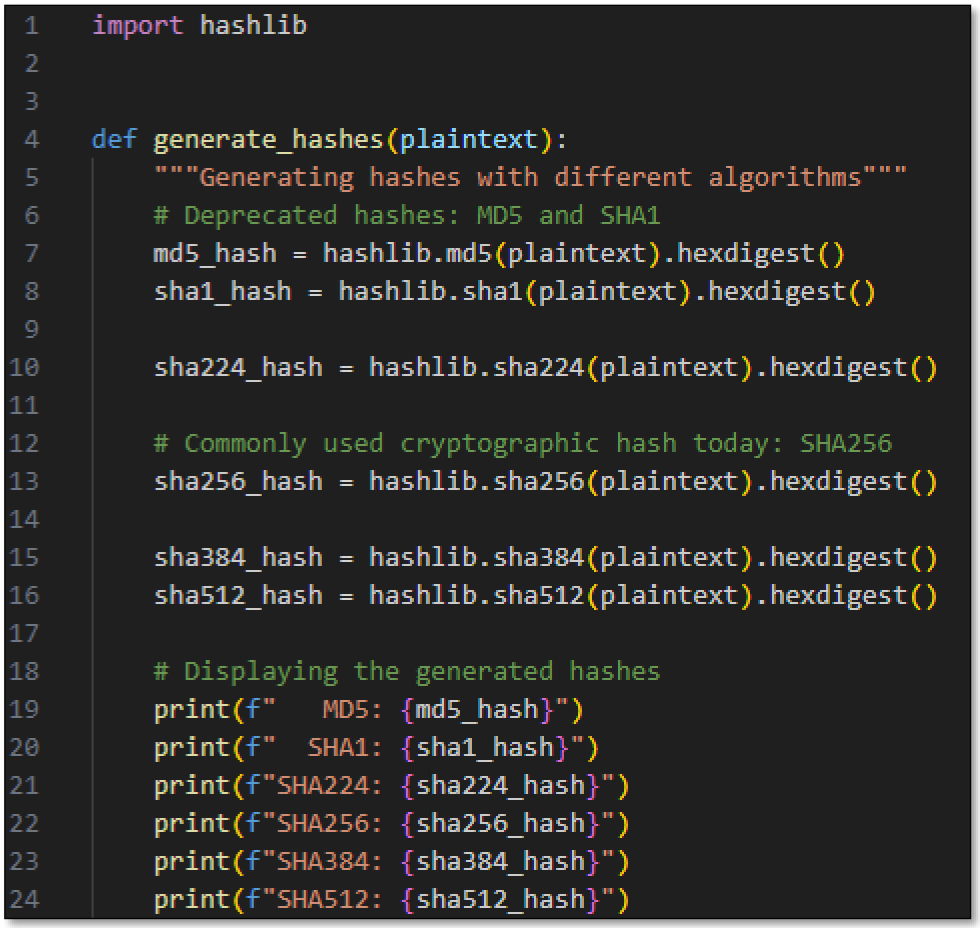


Tutorial 1: Hashing with Python hashlib Library

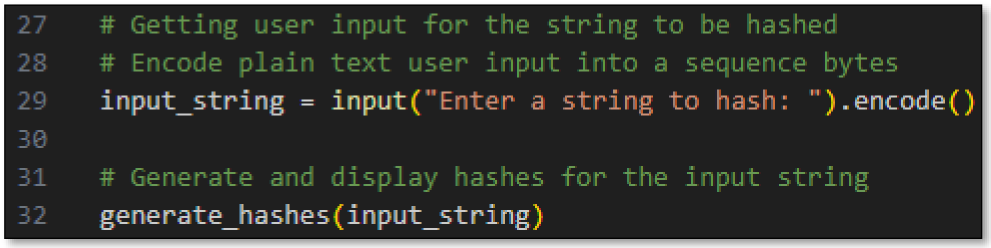
The following program uses several hashing algorithms. SHA256 (256 bits) is the common hash in use today. SHA384 (384 bits) is next in line.

Security is always a tradeoff. The longer the hash is, the more secure it is. It also takes longer to calculate.

Create a Python file named: **hashing.py**

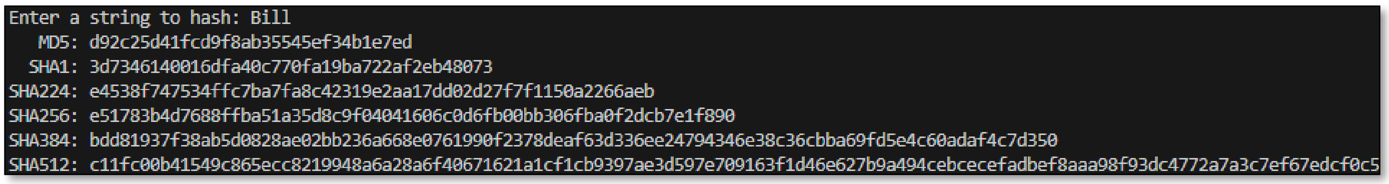


```python
import hashlib


def generate_hashes(plaintext):
    """Generating hashes with different algorithms"""
    # Deprecated hashes: MD5 and SHA1
    md5_hash = hashlib.md5(plaintext).hexdigest()
    sha1_hash = hashlib.sha1(plaintext).hexdigest()

    sha224_hash = hashlib.sha224(plaintext).hexdigest()

    # Commonly used cryptographic hash today: SHA256
    sha256_hash = hashlib.sha256(plaintext).hexdigest()

    sha384_hash = hashlib.sha384(plaintext).hexdigest()
    sha512_hash = hashlib.sha512(plaintext).hexdigest()

    # Displaying the generated hashes
    print(f"   MD5: {md5_hash}")
    print(f"  SHA1: {sha1_hash}")
    print(f"SHA224: {sha224_hash}")
    print(f"SHA256: {sha256_hash}")
    print(f"SHA384: {sha384_hash}")
    print(f"SHA512: {sha512_hash}")
```



```python
# Getting user input for the string to be hashed
# Encode plain text user input into a sequence bytes
input_string = input("Enter a string to hash: ").encode()

# Generate and display hashes for the input string
generate_hashes(input_string)
```

Example run:



```
Enter a string to hash: Bill
   MD5: d92c25d41fcd9f8ab35545ef34b1e7ed
  SHA1: 3d7346140016dfa40c770fa19ba722af2eb48073
SHA224: e4538f747534ffc7ba7fa8c42319e2aa17dd02d27f7f1150a2266aeb
SHA256: e51783b4d7688ffba51a35d8c9f04041606c0d6fb00bb306fba0f2dcb7e1f890
SHA384: bdd81937f38ab5d0828ae02bb236a668e0761990f2378deaf63d336ee24794346e38c36cbba69fd5e4c60adaf4c7d350
SHA512: c11fc00b41549c865ecc8219948a6a28a6f40671621a1cf1cb9397ae3d597e709163f1d46e627b9a494cebcecefadbef8aaa98f93dc4772a7a3c7ef67edcf0c5
```

Notice that no matter how long the word is, the hash lengths stay the same.

How Does the Program Work?

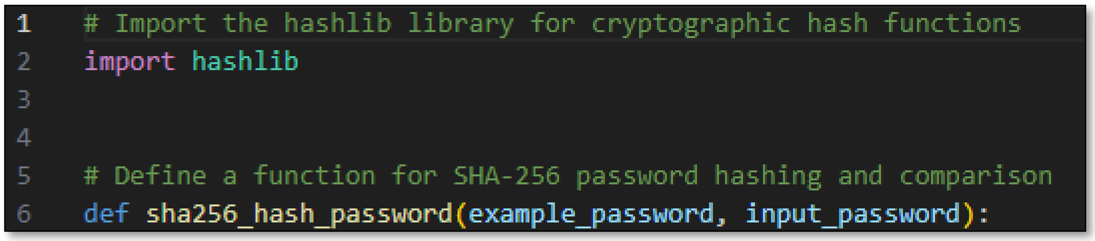
- Import the Python **hashlib** module, which provides cryptographic hashing functionalities.
- The input function takes a plaintext string as input and encodes it into a sequence of bytes.
- The function **generate_hashes()** generates hashes using various hashing algorithms provided by hashlib.
- The **hexdigest()** method converts the byte stream hashes into hexadecimal format for display purposes.

Cryptographic hashing is important, MD5 and SHA1 are considered deprecated due to vulnerabilities and are not recommended for cryptographic purposes. SHA256 is widely used for its stronger security properties. It's crucial to use updated and secure hashing algorithms based on specific security requirements in real-world applications.

Tutorial 2: Hashing with Passwords

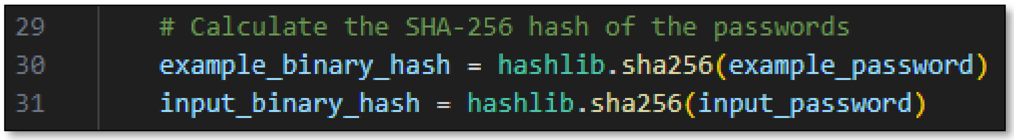
Passwords are never passed in plain text. The password is hashed, the hashes are compared.

Create a Python program named **hashing_password.py**



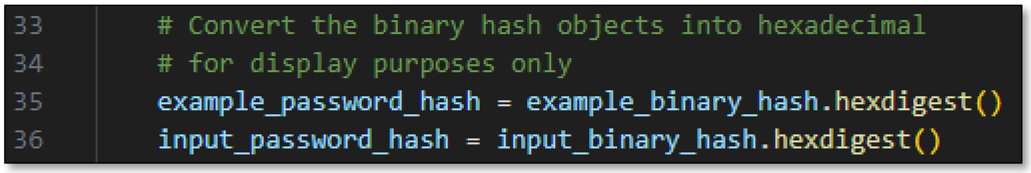
```python
# Import the hashlib library for cryptographic hash functions
import hashlib


# Define a function for SHA-256 password hashing and comparison
def sha256_hash_password(example_password, input_password):
```

We start by importing the hashlib library and defining a function which takes an example password and input password to compare.



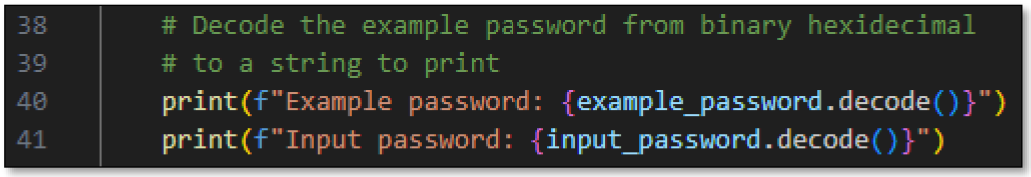
```python
    # Calculate the SHA-256 hash of the passwords
    example_binary_hash = hashlib.sha256(example_password)
    input_binary_hash = hashlib.sha256(input_password)
```

The hashes of the binary versions of the passwords are hashed.

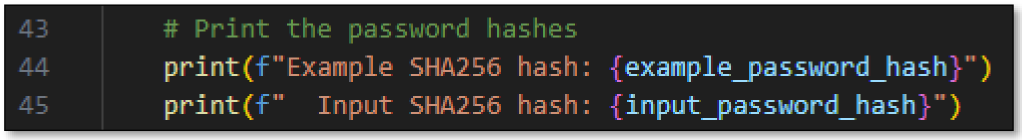


```python
    # Convert the binary hash objects into hexadecimal
    # for display purposes only
    example_password_hash = example_binary_hash.hexdigest()
    input_password_hash = input_binary_hash.hexdigest()
```

The binary hashes are converted into hexadecimal for comparison purposes.

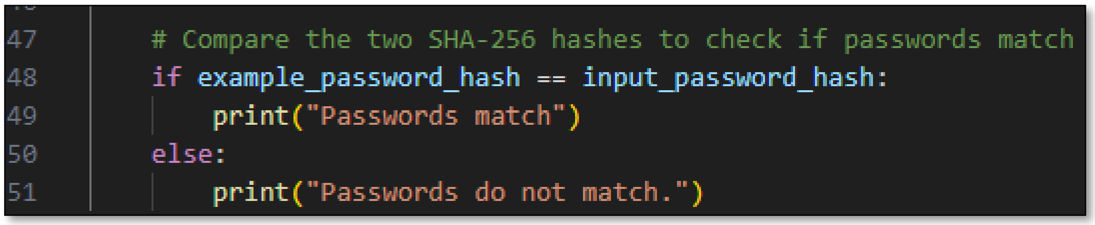


```python
    # Decode the example password from binary hexidecimal
    # to a string to print
    print(f"Example password: {example_password.decode()}")
    print(f"Input password: {input_password.decode()}")
```



```python
    # Print the password hashes
    print(f"Example SHA256 hash: {example_password_hash}")
    print(f"  Input SHA256 hash: {input_password_hash}")
```

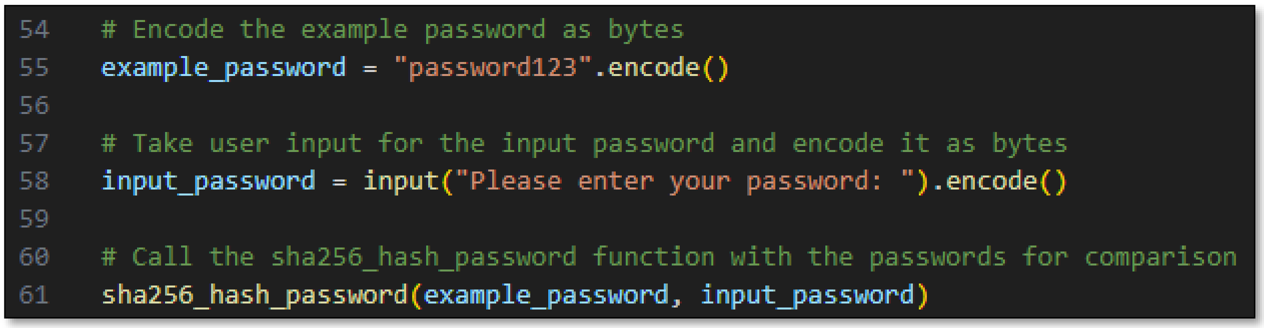
We converting the binary hexadecimal hash to a string for display purposes. A normal hash comparison does not do this. This is for demonstration only.



```python
    # Compare the two SHA-256 hashes to check if passwords match
    if example_password_hash == input_password_hash:
        print("Passwords match")
    else:
        print("Passwords do not match.")
```

We compare the two binary hashes to see if the passwords match. This is what happens when your password is authenticated.

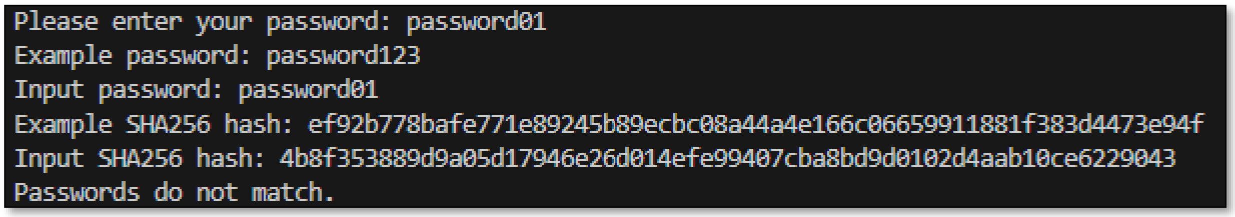
This completes the function. Everything from this point on is aligned to the left.



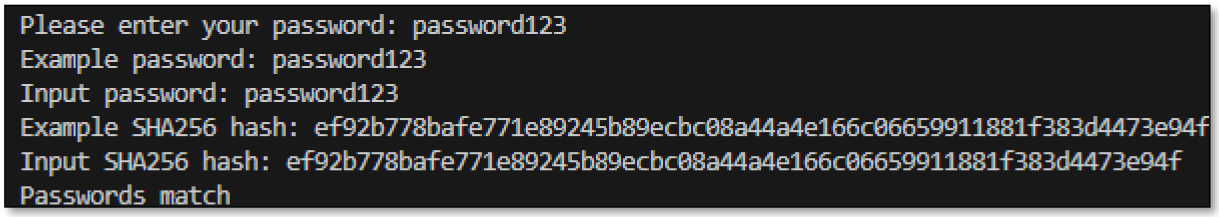
```python
# Encode the example password as bytes
example_password = "password123".encode()

# Take user input for the input password and encode it as bytes
input_password = input("Please enter your password: ").encode()

# Call the sha256_hash_password function with the passwords for comparison
sha256_hash_password(example_password, input_password)
```

The passwords are encoded as a byte stream for hashing purposes and password to the password comparison function.

Example run:



```
Please enter your password: password01
Example password: password123
Input password: password01
Example SHA256 hash: ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f
Input SHA256 hash: 4b8f353889d9a05d17946e26d014efe99407cba8bd9d0102d4aab10ce6229043
Passwords do not match.
```



```
Please enter your password: password123
Example password: password123
Input password: password123
Example SHA256 hash: ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f
Input SHA256 hash: ef92b778bafe771e89245b89ecbc08a44a4e166c06659911881f383d4473e94f
Passwords match
```

Assignment Submission

1. Attach the code.
2. Attach a screenshot showing a successful run of the program.
3. Submit the assignment in Blackboard.