

OEIT1- BCT (2023-2024)

Lab ESE

Date: 06/05/2024

Student Name: Siddhi Dinkar Mardhekar

UCID: 2021200068

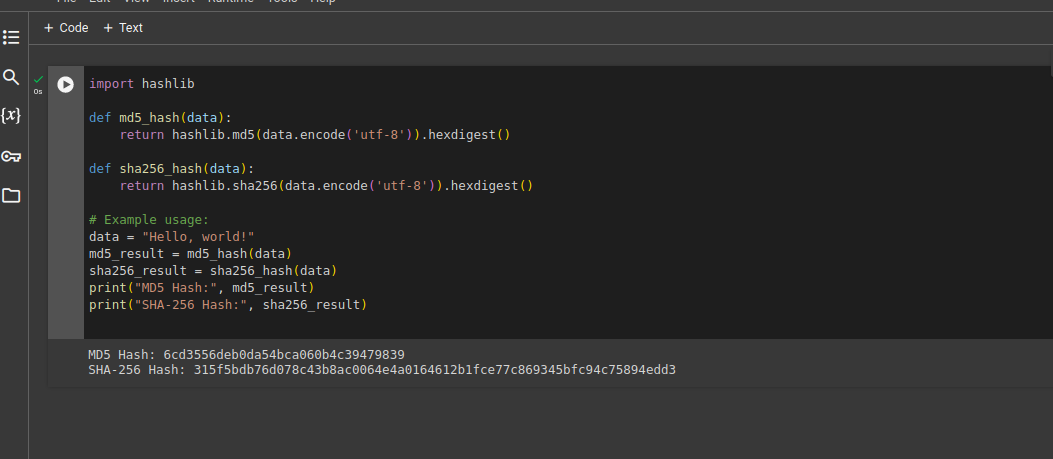
Branch: EXTC

Sem: 6

Lab title: Lab 2: Blockchain Primitives-Cryptosystems

Brief description of each task with screenshots (caption)

Task-1: Perform the cryptographic hash functions of md5 and SHA



This Python code defines two functions, `md5\_hash` and `sha256\_hash`, which compute the MD5 and SHA-256 hashes of a given input string, respectively.

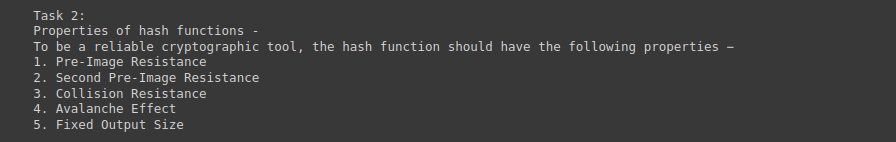
The `md5\_hash` function uses the MD5 hashing algorithm from the `hashlib` library to generate the MD5 hash of the input data, while the `sha256\_hash` function utilizes the SHA-256 hashing algorithm for the same purpose.

These functions take a string `data` as input, encode it in UTF-8 format (required by hashlib), compute the hash using the specified algorithm, and return the hexadecimal representation of the hash.

The example usage demonstrates how to call these functions with a sample input string ("Hello, world!"), compute the respective MD5 and SHA-256 hashes, and print the results.

This code provides a simple and straightforward way to generate cryptographic hashes using two commonly used hashing algorithms in Python.

Task-2: Demonstrate properties of cryptographic hash functions.



1. Pre-image Resistance: Given a hash value h, it should be computationally infeasible to find any input m such that h = hash(m). In other words, it should be difficult to reverse the hashing process and find the original input from the hash value alone.

2. Second Pre-image Resistance: Given an input m1, it should be computationally infeasible to find another input m2 ≠ m1 such that hash(m1) = hash(m2). This property ensures that it's difficult to find a different input with the same hash value as a given input.

3. Collision Resistance: It should be computationally infeasible to find any two distinct inputs m1 and m2 that produce the same hash value (i.e., hash(m1) = hash(m2)). This property ensures that it's challenging to find collisions, which helps prevent attacks like birthday attacks.

4. Avalanche Effect: A small change in the input should produce a significant change in the output hash value. This property ensures that similar inputs produce vastly different hash values, adding randomness and making it difficult for attackers to predict or manipulate the hash output.

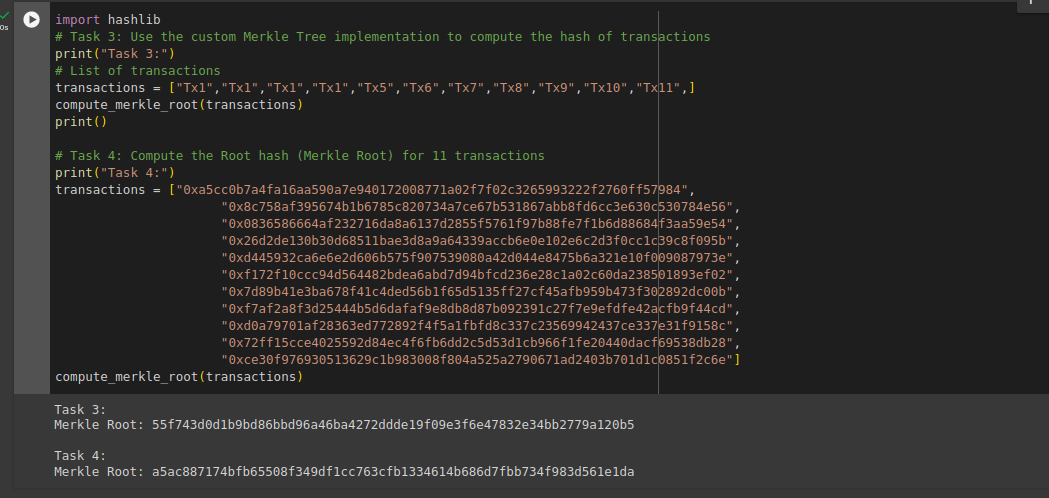
5. Fixed Output Size: Hash functions produce output of a fixed size, regardless of the input size. This property makes them suitable for applications where a fixed-length representation of data is required, such as digital signatures and checksums.

These properties collectively make cryptographic hash functions crucial for a wide range of applications, including password hashing, digital signatures, data integrity verification, and more.

Task-3: Use the pymerkel library to compute the hash of transaction

and

Task-4:Compute the root hash (Merkle root) for 11 transactions.



* `import hashlib` is a Python statement that imports the `hashlib` library/module.
* The `hashlib` library provides a convenient interface to various secure hash and message digest algorithms, including MD5, SHA-1, SHA-224, SHA-256, SHA-384, and SHA-512.
* These algorithms are commonly used for cryptographic operations, such as generating hash values for data integrity verification, password hashing, digital signatures, and more.
* Once imported, you can use functions from the `hashlib` module to compute hash values using these algorithms, as demonstrated in the code snippets provided.

Lab ese Setup:

