Information Security

Assignment # 05

Name: Hamza Shahid
Roll No: 20P-0117
SE-7A

Merkle-Damgard Construction:

The Merkle Damgard construction is a process of making a cryptographic hash function using a one-way compression function. This construction is based on the rule that if the compression function is collision resistance, the hash function will also be collision resistance. Many popular hash functions like **MD5**, **SHA-1**, and **SHA-2** have been designed using Merkle Damgard construction.

Working:

The process starts with expanding the input message to a length that is multiple of some fixed number of bits. It is necessary because the compression function only works on the fixed-length inputs.

Why padding is important:

It is important to carefully select the padding scheme for message length expansion because weak padding can introduce security vulnerability to the function. Padding should be **MD-compliant** which means it should satisfy following conditions:

- M is prefix of pad(M)
- If M1 == M2 then pad(M1) == pad(M2)
- If M1 != M2 then the last block of pad(M1) != last block of pad(M2)

Code:

```
import hashlib

# Define the compression function (SHA-256 in this example)
def sha256_compress(block, state):
    sha256 = hashlib.sha256()
    sha256.update(block + state)
    return sha256.digest()

# Implement the Merkle-Damgård construction
def merkle_damgard(message, initial_state, block_size):
    state = initial_state

    # Pad the message to a multiple of the block size
    message += b'\x80'  # Append 1 bit
    while len(message) % block_size != (block_size - 8):
        message += b'\x00'  # Append 0 bits

    # Append the message length as a 64-bit big-endian integer
```

```
    message += (8 * len(message)).to_bytes(8, byteorder='big')

    # Process the message in blocks
    for i in range(0, len(message), block_size):
        block = message[i:i + block_size]
        state = sha256_compress(block, state)

    return state

# Example usage
if __name__ == "__main__":
    message = b"Hello, Merkle-Damgard!"
    initial_state = hashlib.sha256(b"Initial state").digest()
    block_size = 64  # SHA-256 block size in bytes

    result = merkle_damgard(message, initial_state, block_size)

    print("Hash result:", result.hex())
```

Explanation:

In this example:

- We define a compression function, sha256_compress, which simulates the behavior of the SHA-256 compression function. You should replace this with your custom compression function if needed.
- The merkle_damgard function performs the Merkle-Damgård construction. It pads the message to the appropriate length, breaks it into blocks, and iteratively compresses each block using the compression function.
- We provide an example message, an initial state, and the block size. The result is the hash value of the message based on the Merkle-Damgård construction.