Collision Attack on the Concatenation Combiner

In this exercise, we consider Merkle-Damgård hash functions. For simplicity we do not use any padding (but all of this exercise would apply as well if we used a secure padding scheme).

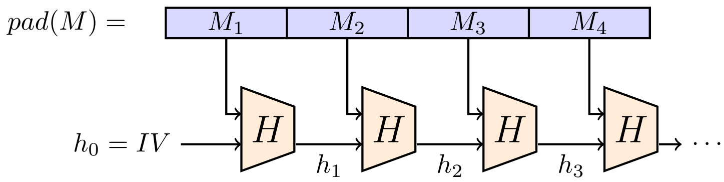

Figure 1: Merkle-Damgård construction.

The *concatenation combiner* combines the output of two Merkle-Damgård hashes applied to the same message. These two hash functions F_1, F_2 use different IVs and different compression functions f_1 and f_2. The output of the concatenation combiner is:

$$F(m_0, \ldots, m_{t-1}) = F_1(m_0, \ldots, m_{t-1}) \| F_2(m_0, \ldots, m_{t-1}) \ .$$

We suppose that the message blocks, the chaining values and the outputs of both functions have n bits each. Thus the combiner outputs $2n$ bits.

Question 1. *Show that by using messages of length $n/2$ blocks, one can build a $2^{n/2}$-multicollision of F_1 in time $\mathcal{O}(n2^{n/2})$, i.e., a set of messages x_i having all the same length and the same output chaining value. How much space do you need to store this multicollision?*

Question 2. *Show that we can find a collision of F in time $\mathcal{O}(n2^{n/2})$.*

We give the definition of our two MD hash functions. Both of them take as input a list of 64-bit integers (the message blocks) and return a 32-bit integer. Recall that you can use the function `randrange(1 << 64)` to output a random 64-bit integer.

```python
from hashlib import sha256, md5
from random import randrange


def md_hash_1(message_blocks, initial_value=0x12345678):
    """
    Args:
        message_blocks: list of message blocks as 64-bit integers
        initial_value (int): initial value as 32-bit integer

    Returns:
        int: hash of the message as a 32-bit integer.
    """
    chaining_value = initial_value

    for block in message_blocks:
        data = chaining_value.to_bytes(4, 'little') + block.to_bytes(8, 'little')
```

```python
        chaining_value = int(sha256(data).hexdigest()[:8], 16)  # Use first 32 bits

    return chaining_value


def md_hash_2(message_blocks, initial_value=0x87654321):
    chaining_value = initial_value

    for block in message_blocks:
        data = chaining_value.to_bytes(4, 'little') + block.to_bytes(8, 'little')
        chaining_value = int(md5(data).hexdigest()[:8], 16)  # Use first 32 bits

    return chaining_value
```

Question 3. *Implement the collision attack on the concatenation combiner of* `md_hash_1` *and* `md_hash_2`*: find a pair of messages m_1, m_2 such that both hash functions have the same output.*

Preimage Attack on the Concatenation Combiner

Question 4. *Show that given a target t, we can find a set of $\mathcal{O}(2^n)$ preimages of t by F_1 in time $\mathcal{O}(2^n)$.*

Question 5. *Deduce that we can find preimages of the concatenation combiner in time $\mathcal{O}(2^n)$.*

We now use two different MD functions with 20-bit output.

```python
def md_hash_3(message_blocks, initial_value=0x12345678):
    chaining_value = initial_value
    for block in message_blocks:
        data = chaining_value.to_bytes(4, 'little') + block.to_bytes(8, 'little')
        chaining_value = int(sha256(data).hexdigest()[:5], 16)
    return chaining_value


def md_hash_4(message_blocks, initial_value=0x87654321):
    chaining_value = initial_value
    for block in message_blocks:
        data = chaining_value.to_bytes(4, 'little') + block.to_bytes(8, 'little')
        chaining_value = int(md5(data).hexdigest()[:5], 16)
    return chaining_value
```

Question 6. *Find a preimage of 0,0.*

Question 7 (Bonus question)**.** *Go back to the two first MD hashes. Append a third MD hash, this time using sha1 and 32 bits of output. Find a collision of the triple concatenation combiner. What is the number of blocks of the colliding messages?*