Blake3 Hash Scheme Documentation

What is BLAKE3?

BLAKE3 is a cryptographic hash function that is fast, widely supported, and does not have any variants unlike its previous version Blake2. Blake2 had many variants: Blake2bp, Blake2sp etc. While Blake2b was widely supported, it was not fast and Blake2bp, Blake2sp were fast, but not widely supported. Blake3 solves all these drawbacks.

Overview of the Implementation

Functions:

1. quarter_round
2. blake3_compress
3. blake3_hash

1. quarter_round(a, b, c, d)

The quarter_round function is a main component of the hash function's mixing process. It takes four integers and performs a series of operations to mix them.

Parameters:

- a, b, c, d: Four integers to be mixed.

Returns:

- A tuple of four integers after mixing.

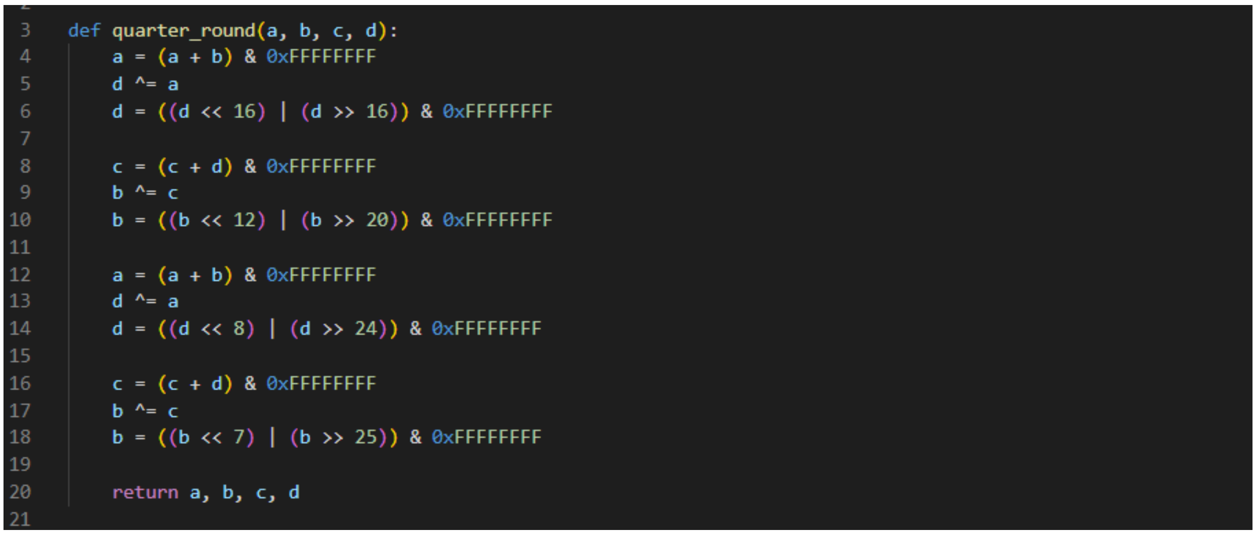
Explanation:

- The function adds, XORs, and rotates the bits of the input integers to ensure thorough mixing.

Example Usage:

a, b, c, d = quarter_round(1, 2, 3, 4)

Preview:



```python
def quarter_round(a, b, c, d):
    a = (a + b) & 0xFFFFFFFF
    d ^= a
    d = ((d << 16) | (d >> 16)) & 0xFFFFFFFF

    c = (c + d) & 0xFFFFFFFF
    b ^= c
    b = ((b << 12) | (b >> 20)) & 0xFFFFFFFF

    a = (a + b) & 0xFFFFFFFF
    d ^= a
    d = ((d << 8) | (d >> 24)) & 0xFFFFFFFF

    c = (c + d) & 0xFFFFFFFF
    b ^= c
    b = ((b << 7) | (b >> 25)) & 0xFFFFFFFF

    return a, b, c, d
```

2. blake3_compress(state)

The blake3_compress function applies the quarter_round function repeatedly to compress the state.

Parameters:

- state: A list of integers representing the state.

Returns:

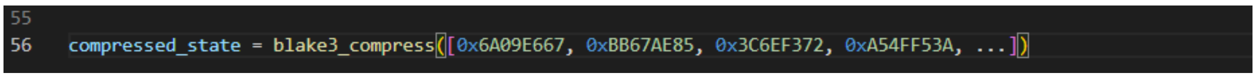
- A list of integers after compression.

Explanation:

- The function performs 12 rounds of mixing using the quarter_round function on different parts of the state.

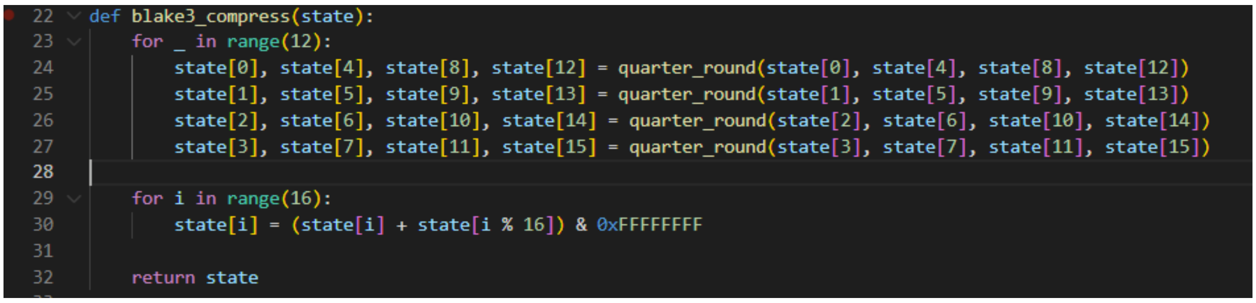
- It then adds the initial state to the final state to mix in the original values.

Example Usage:



```python
compressed_state = blake3_compress([0x6A09E667, 0xBB67AE85, 0x3C6EF372, 0xA54FF53A, ...])
```

Preview:



```python
def blake3_compress(state):
    for _ in range(12):
        state[0], state[4], state[8], state[12] = quarter_round(state[0], state[4], state[8], state[12])
        state[1], state[5], state[9], state[13] = quarter_round(state[1], state[5], state[9], state[13])
        state[2], state[6], state[10], state[14] = quarter_round(state[2], state[6], state[10], state[14])
        state[3], state[7], state[11], state[15] = quarter_round(state[3], state[7], state[11], state[15])

    for i in range(16):
        state[i] = (state[i] + state[i % 16]) & 0xFFFFFFFF

    return state
```

3. blake3_hash(key, counter, message)

The blake3_hash function initializes the state with a key and counter, compresses the state, and produces a hash output.

Parameters:

- key: A byte string used as the key for hashing.

- counter: A counter value for domain separation.

- message: The message to be hashed.

Returns:

- A byte string representing the hash output.

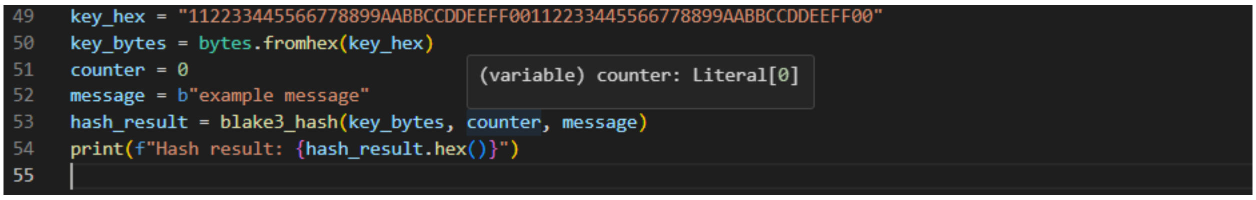
Explanation:

- The function initializes the state with predefined constants, the key, and the counter.

- It calls blake3_compress to process the state.

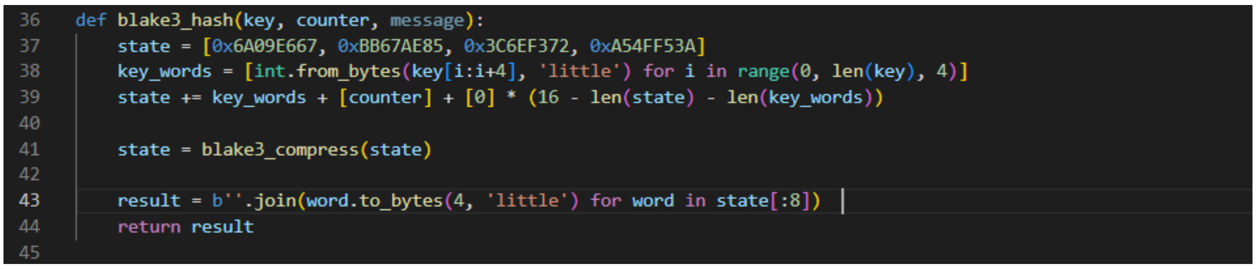
- Finally, it converts the state to a byte string to produce the hash output.

Example Usage:



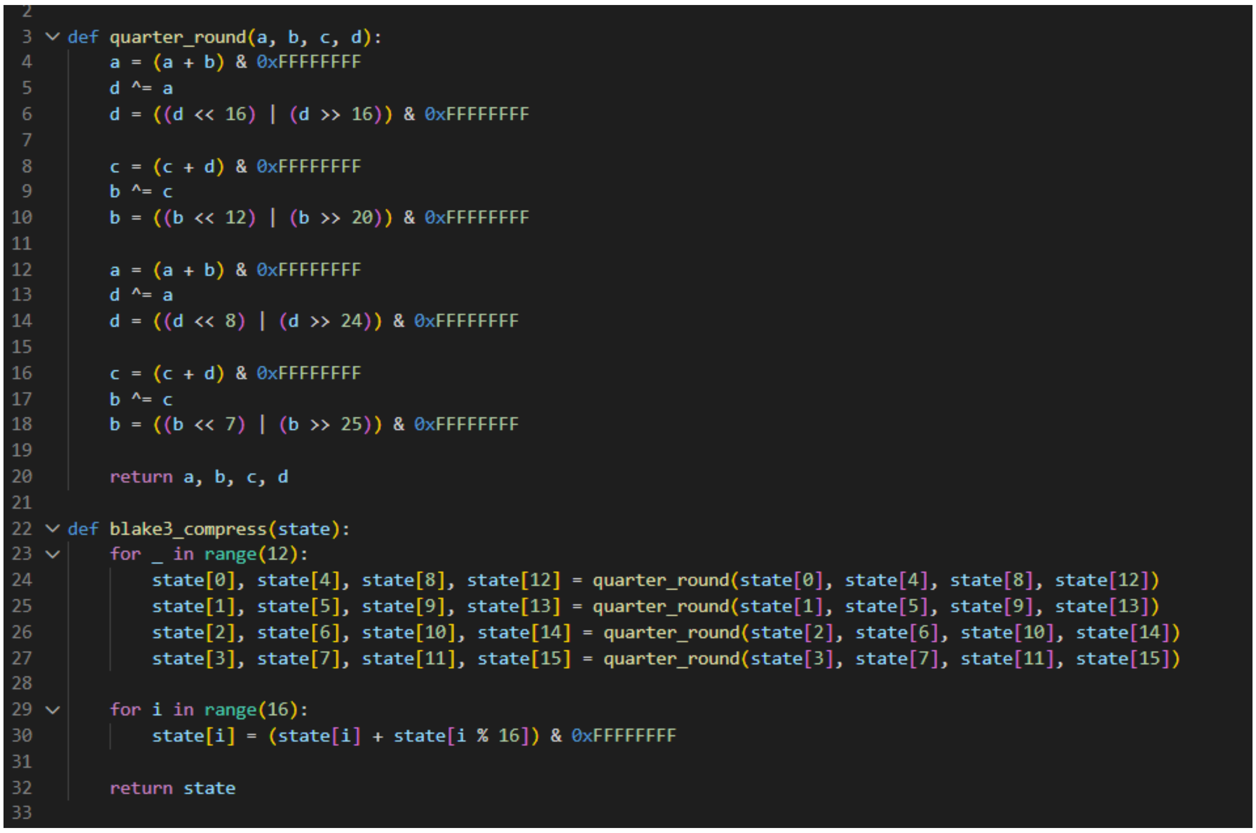
```python
key_hex = "112233445566778899AABBCCDDEEFF00112233445566778899AABBCCDDEEFF00"
key_bytes = bytes.fromhex(key_hex)
counter = 0
message = b"example message"
hash_result = blake3_hash(key_bytes, counter, message)
print(f"Hash result: {hash_result.hex()}")

```

Preview:

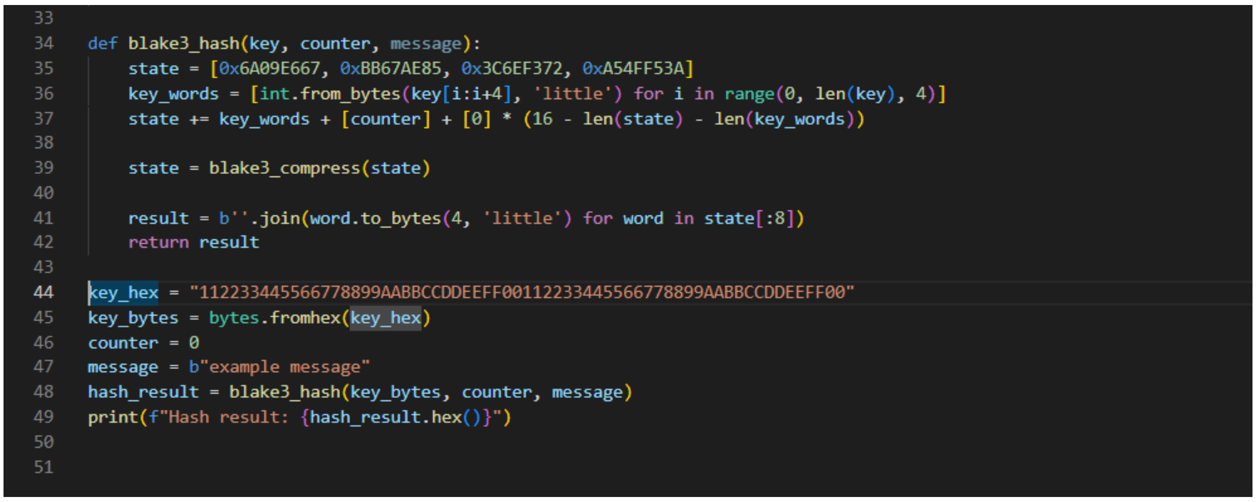


```python
def blake3_hash(key, counter, message):
    state = [0x6A09E667, 0xBB67AE85, 0x3C6EF372, 0xA54FF53A]
    key_words = [int.from_bytes(key[i:i+4], 'little') for i in range(0, len(key), 4)]
    state += key_words + [counter] + [0] * (16 - len(state) - len(key_words))

    state = blake3_compress(state)

    result = b''.join(word.to_bytes(4, 'little') for word in state[:8])
    return result

```

The whole code:



```python

def quarter_round(a, b, c, d):
    a = (a + b) & 0xFFFFFFFF
    d ^= a
    d = ((d << 16) | (d >> 16)) & 0xFFFFFFFF

    c = (c + d) & 0xFFFFFFFF
    b ^= c
    b = ((b << 12) | (b >> 20)) & 0xFFFFFFFF

    a = (a + b) & 0xFFFFFFFF
    d ^= a
    d = ((d << 8) | (d >> 24)) & 0xFFFFFFFF

    c = (c + d) & 0xFFFFFFFF
    b ^= c
    b = ((b << 7) | (b >> 25)) & 0xFFFFFFFF

    return a, b, c, d

def blake3_compress(state):
    for _ in range(12):
        state[0], state[4], state[8], state[12] = quarter_round(state[0], state[4], state[8], state[12])
        state[1], state[5], state[9], state[13] = quarter_round(state[1], state[5], state[9], state[13])
        state[2], state[6], state[10], state[14] = quarter_round(state[2], state[6], state[10], state[14])
        state[3], state[7], state[11], state[15] = quarter_round(state[3], state[7], state[11], state[15])

    for i in range(16):
        state[i] = (state[i] + state[i % 16]) & 0xFFFFFFFF

    return state

```



```python

def blake3_hash(key, counter, message):
    state = [0x6A09E667, 0xBB67AE85, 0x3C6EF372, 0xA54FF53A]
    key_words = [int.from_bytes(key[i:i+4], 'little') for i in range(0, len(key), 4)]
    state += key_words + [counter] + [0] * (16 - len(state) - len(key_words))

    state = blake3_compress(state)

    result = b''.join(word.to_bytes(4, 'little') for word in state[:8])
    return result

key_hex = "112233445566778899AABBCCDDEEFF00112233445566778899AABBCCDDEEFF00"
key_bytes = bytes.fromhex(key_hex)
counter = 0
message = b"example message"
hash_result = blake3_hash(key_bytes, counter, message)
print(f"Hash result: {hash_result.hex()}")


```