(a) Algorithm Description

1 It extracts the SET command from the XDP payload.

2 SET command founded, try to figure the index of the Memcached key in payload.

3 If the key is found, process the key with hash function.

4 If the hash matches the calculated hash, the corresponding cache entry in the map should be invalidated

(b) REX Implementation 🟩

```
1  let set_iter = payload                                   1
2   .windows(4) // 4 chars as a slice
3   .enumerate()
4   .filter_map(|(i, v)|
5      if v == b"set " { Some(i) } else { None }
6  ); // found the SET command
7  for index in set_iter {                                 2
8    ... // set payload index via SET command
9    payload                                               3
10    .iter()
11    .take_while(|&&c| c != b' ')
12    .for_each(|&c| {
13      ... // process the key with hash func
14    });
15  ... // invalidate Memcached cache entry                 4
16}     // if the hash matches
```

(c) Original BMC Implementation 🟦

```
1 #pragma clang loop unroll(disable)                        1
2 for (unsigned int off = 0;
3   off < BMC_MAX_PACKET_LENGTH &&
4   payload + off + 1 <= data_end; off++) {
5   if (set_found == 0 && payload[off] == 's' &&
6      payload + off + 3 <= data_end &&
7      payload[off + 1] == 'e' &&
8      payload[off + 2] == 't') {
9     ... // move offset after the SET command
10    set_found = 1;
11  } else if (key_found == 0 && set_found == 1            2
12             && payload[off] != ' ') {
13    if (payload[off] == '\r') {
14      set_found = 0;
15      key_found = 0;
16    } else {
17      ... // found the start of the key
18      key_found = 1;
19    }
20  } else if (key_found == 1) {                           3
21    if (payload[off] == ' ') {
22      ... // found the end of the key
23      set_found = 0;
24      key_found = 0;
25    } else {
26      ... // process the key with hash func
27    }     // invalidate Memcached cache entry            4
28  }       // if the hash matches
29}
```

- The checks required previously by the verifier, including these for offset and data_end limits (**L6**), are now being enforced via the inherent language features of Rust, such as the slice that implements bound checks (**L2**) and (**L10**).

The four levels of nesting (**L5,11,20**), is significantly reduced by converting a `for`-loop (**L2**), with intricate conditions (**L4**) into a clean chain of higher-order functions with closures through the take_while (**L11**), which will filter the Memcached SET key (**L5**) from the payload with the iterator generated by filter_map (**L4**), thus dividing the code into three distinct sequential parts.