auth_hmacsha256.c

Function 1

```
size_t crypto_auth_hmacsha256_bytes(void) {
①{  return crypto_auth_hmacsha256_BYTES;
}
```

→CFG:

→Term Requirements:

TR = { }

→Test Paths:

TP = [1]

→Inputs : None

Function 2

```
size_t crypto_auth_hmacsha256_keybytes(void) {
①{ return crypto_auth_hmacsha256_KEYBYTES;
}
```

TR = { } , TP = [1] , Inputs = None

Function 3

```
size_t crypto_auth_hmacsha256_statebytes(void) {
①{  return sizeof(crypto_auth_hmacsha256_state);
}
```

TR = { } , TP = [1] , Inputs : None

Function 4

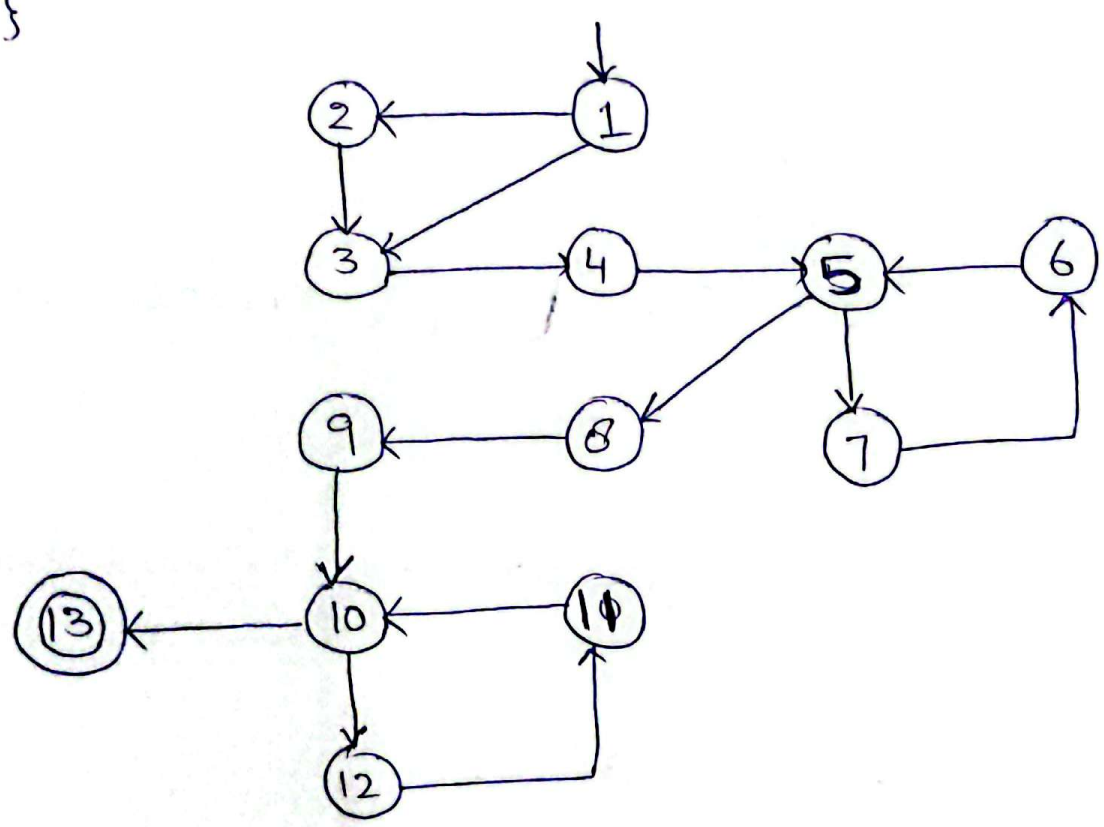
```
void crypto_auth_hmacsha256_keygen(unsigned char k[crypto_auth_hmacsha256_KEYBYTES])
    randombytes(k, crypto_auth_hmacsha256_KEYBYTES);
}
```

TR = { } , TP = [1] , Inputs : Any

Function 5

```
int crypto_auth_hmacsha256_init(crypto_auth_hmacsha256_state *state,
                                const unsigned char *key, size_t keylen) {
(1) unsigned char pad[64];
    unsigned char khash[32];
    size_t i;
    if (keylen > 64) {
(2)     crypto_hash_sha256_init(&state->ictx);
        crypto_hash_sha256_update(&state->ictx, key, keylen);
        crypto_hash_sha256_final(&state->ictx, khash);
        key = khash;
        keylen = 32;
    }
(3) crypto_hash_sha256_init(&state->ictx);
    memset(pad, 0x36, 64);
    for (i = 0; i < keylen; i++) {
    (4) (5) (6)
    (7) pad[i] ^= key[i];
    }
```

```
(8)  crypto_hash_sha256_update(&state->ictx, pad, 64);
     crypto_hash_sha256_init(&state->octx);
     memset(pad, 0x5c, 64);
     for (i=0; i<keylen; i++) {
(12)     pad[i] ^= key[i];
     }
(13) crypto_hash_sha256_update(&state->octx, pad, 64);
     sodium_memzero((void *) pad, sizeof pad);
     sodium_memzero((void *) khash, sizeof khash);
     return 0;
}
```



$TR = \{(1,2)^{(1,3)}, (2,3), (3,4), (4,5), (5,7), (5,8), (7,6), (6,5), (8,9), (9,10), (10,12), (10,13), (12,11), (11,10)\}$

$TP_1 = [1, 2, 3, 4, 5, 7, 6, 5, 8, 9, 10, 12, 11, 10, 13]$

$TP_2 = [1, 3, 4, 5, 8, 9, 10, 13]$

Inputs:

→ For TP_2, the "keylen" = 0. parameter.

→ For TP_1, no combination of inputs can achieve this test path. Because both for loops will iterate more than 1 time.

Function 6

```
int crypto_auth_hmacsha256_update(crypto_auth_hmacsha256_state
                *state, const unsigned char *in, unsigned long long inlen){
(1) [ crypto_hash_sha256_update(&state->ictx, in, inlen);
      return 0;
}
```

TR = { }, TP = [1], Inputs: Any

Function 7

```
int crypto_auth_hmacsha256_final(crypto_auth_hmacsha256_state *state,
                unsigned char *out) {
(1) [ unsigned char ihash[32];
      crypto_hash_sha256_final(&state->ictx, ihash);
      crypto_hash_sha256_update(&state->octx, ihash, 32);
      crypto_hash_sha256_final(&state->octx, out);
      sodium_memzero((void *) ihash, sizeof ihash);
      return 0;
}
```

TR = { }, TP = [1], Inputs: Any

Function 8

```
int crypto_auth_hmacsha256(unsigned char *out, const unsigned
    char *in, unsigned long long inlen, const unsigned char *k) {
①   crypto_auth_hmacsha256_state state;
    crypto_auth_hmacsha256_init(&state, k, crypto_auth_hmacsha256_KEYBYTES);
    crypto_auth_hmacsha256_update(&state, in, inlen);
    crypto_auth_hmacsha256_final(&state, out);
    return 0;
}
```

TR = { } , TP = [1] , Inputs : Any

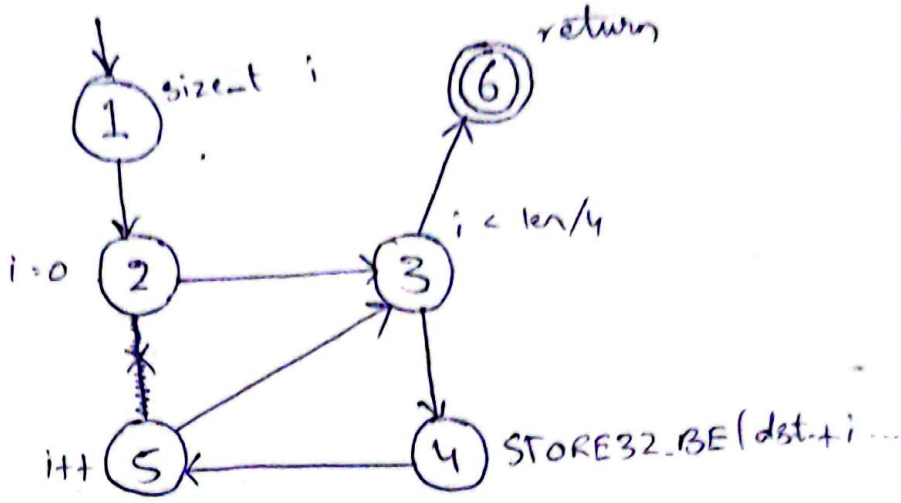
Function 9

```
int crypto_auth_hmacsha256_verify(const unsigned char *h, const unsigned
    char *in, unsigned long long inlen, const unsigned char *k) {
①   unsigned char correct[32];
    crypto_auth_hmacsha256(correct, in, inlen, k);
    return crypto_verify_32(h, correct) | (-(h == correct)) |
           sodium_memcmp(correct, h, 32);
}
```

TR = { } , TP : [1] , Inputs Any

hash_sha256_cp.c

Function 1



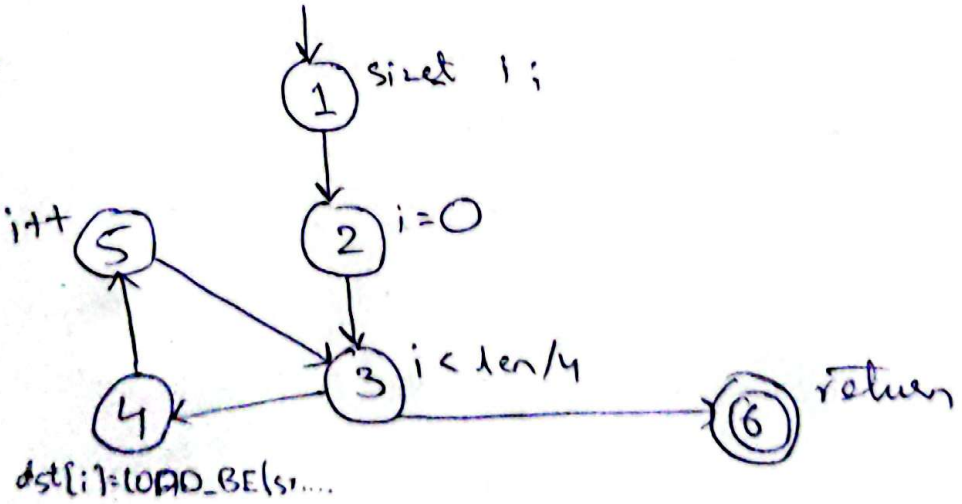
$TR = \{(1,2),(2,3),(3,4),(4,5),(5,3),(3,6)\}$

$TP = [1,2,3,4,5,3,6]$

Inputs:

i) len = 4

Function 2



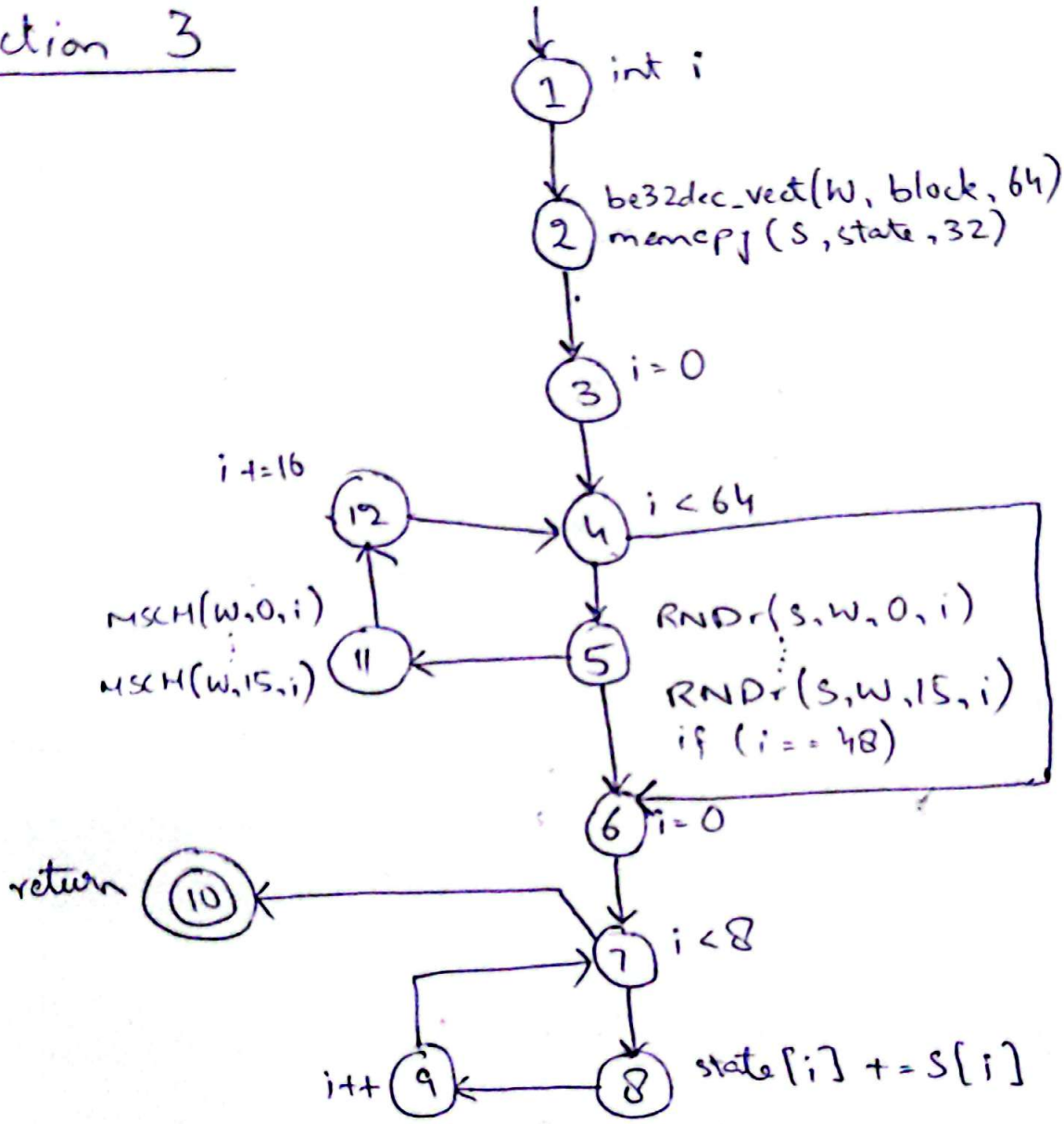
$TR = \{(1,2),(2,3),(3,4),(4,5),(5,3),(3,6)\}$

$TP = [1,2,3,4,5,3,6]$

Inputs:

i) len = 4

Function 3



$TR = \{(1,2),(2,3),(3,4),(4,5),(5,11),(11,12),(12,4),(4,6),(5,6),(6,7),(7,8),(8,9),(9,7),(7,10)\}$

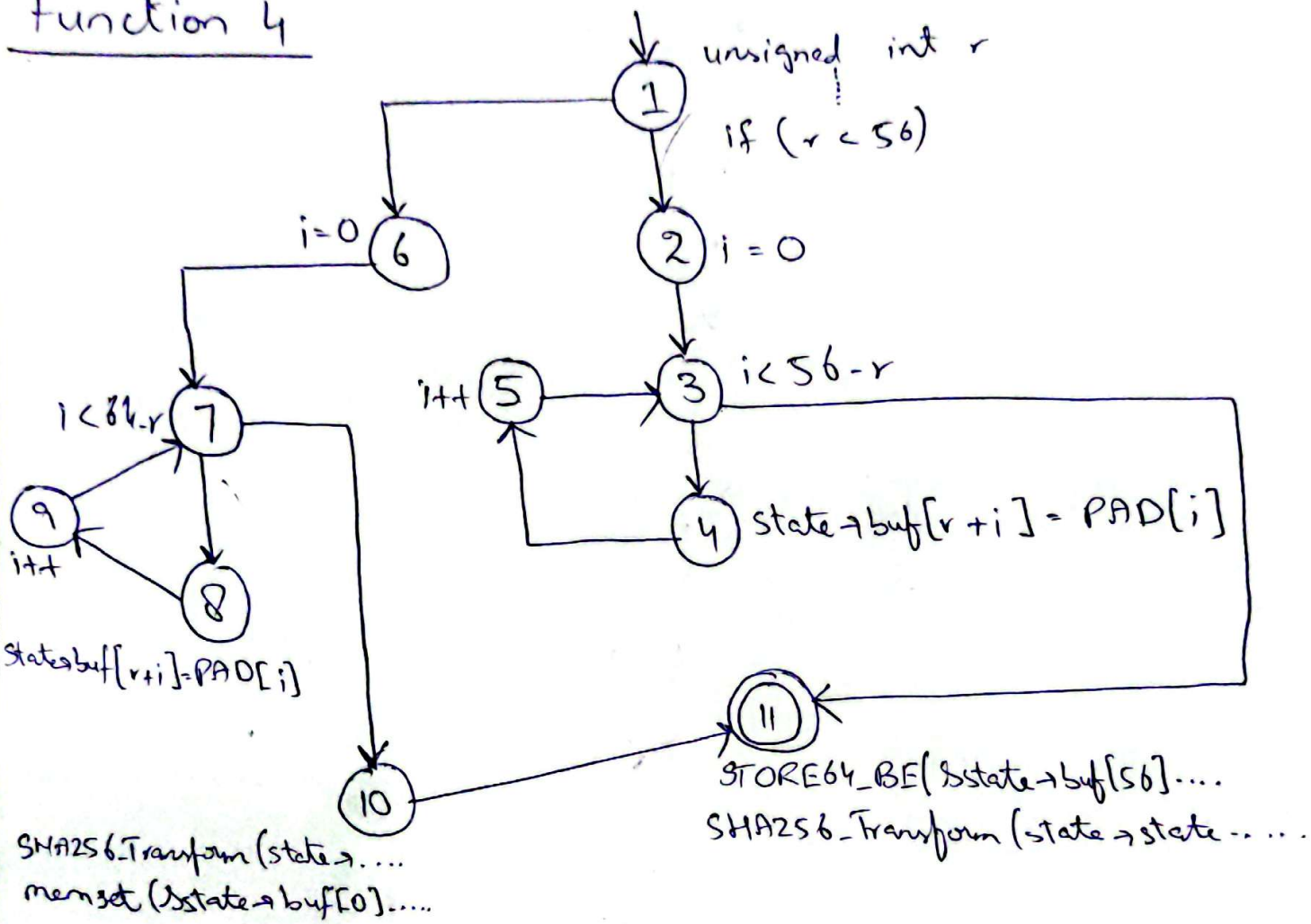
$TP_1 = [1,2,3,4,5,11,12,4,6,7,8,9,7,10]$

$TP_2 = [1,2,3,4,5,6,7,10]$

Inputs:

TP_1	TP_2
→ None (Cannot be achieved)	→ None (Cannot be achieved)

Function 4



$$TR = \{(1,2),(2,3),(3,4),(4,5),(5,3),(3,11),(1,6),(6,7),(7,8),(8,9),(9,7),(7,10),(10,11)\}$$

$$TP_1 = [1,2,3,4,5,3,11]$$

$$TP_2 = [1,6,7,8,9,7,10,11]$$

Inputs:

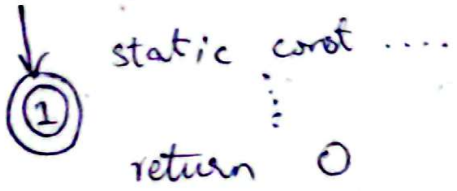
TP_1

1) Set state->count = 440

TP_2

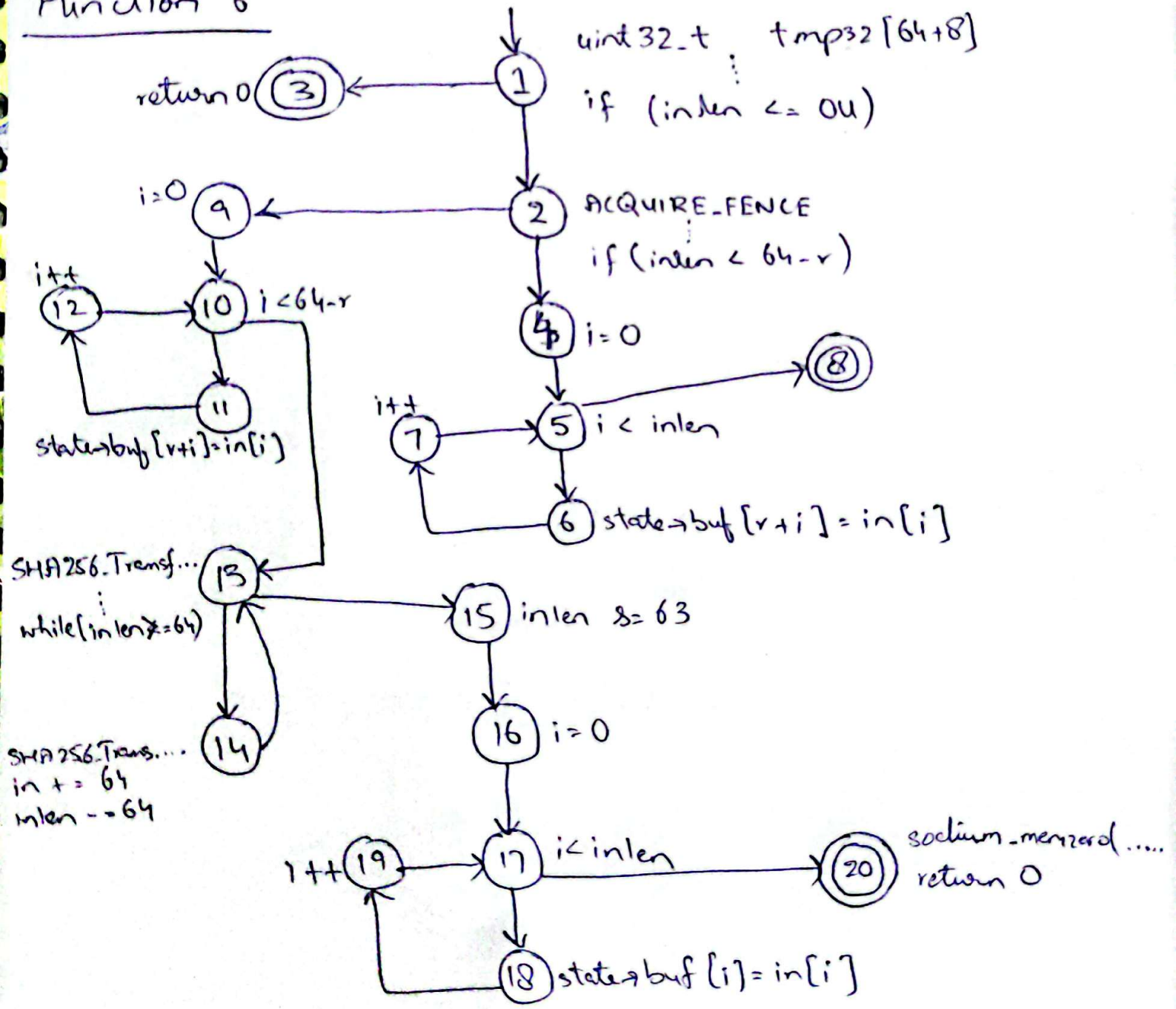
1) Set state->count = 504

Function 5



TR = { } , TP = [1] , Inputs: Any

Function 6



$TR = \{(1,3),(1,2),(2,4),(4,5),(5,6),(6,7),(7,5),(5,8),$
$(2,9),(9,10),(10,11),(11,12),(12,10),(10,13),(13,14),$
$(14,13),(13,15),(15,16),(16,17),(17,18),(18,19),(19,17),(17,20)\}$

$TP_1 = [1, 3]$

$TP_2 = [1, 2, 4, 5, 6, 7, 5, 8]$

$TP_3 = [1, 2, 9, 10, 11, 12, 10, 13, 14, 13, 15, 16, 17, 18, 19, 17, 20]$

Inputs:

TP_1

1) Set inlen = 0

TP_2

Cannot be achieved since the first for loop has to iterate atleast 2 times.

TP_3

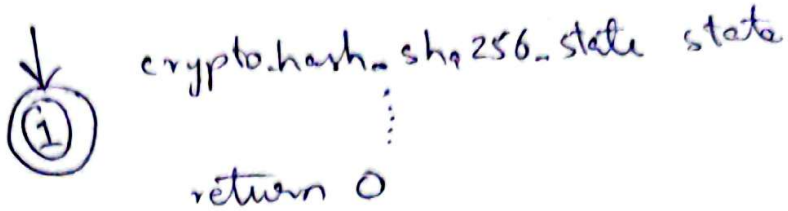
1) Set state->count = 504, in = atleast 66 bytes of data and inlen = 66.

Function 7


```
uint32_t tmp32[64+8]
⋮
return 0
```

$TR = \{\}$, $TP = [1]$, Inputs: Any

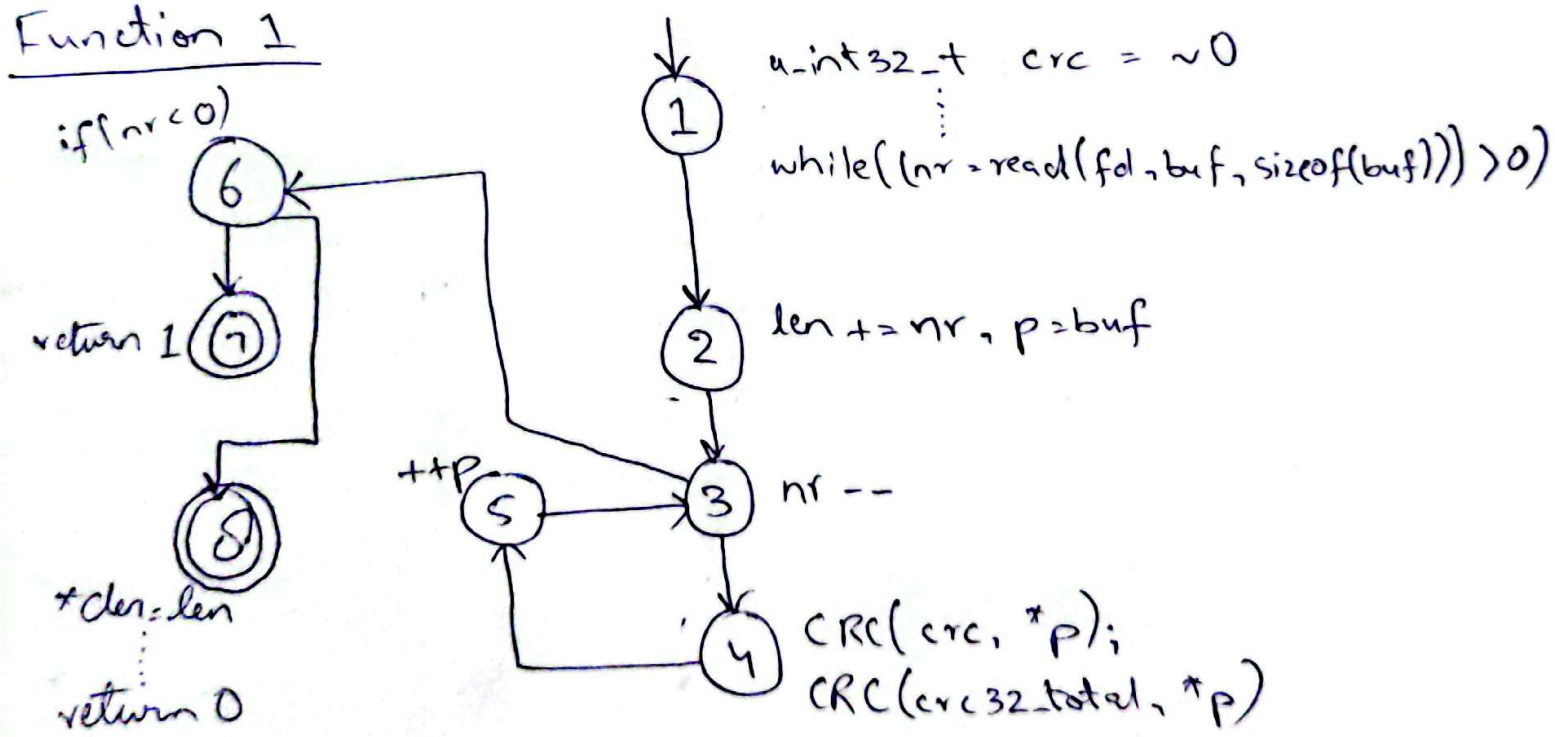
Function 8:



TR = { } , TP = [1] , Inputs : Any

CRC32.c

Function 1



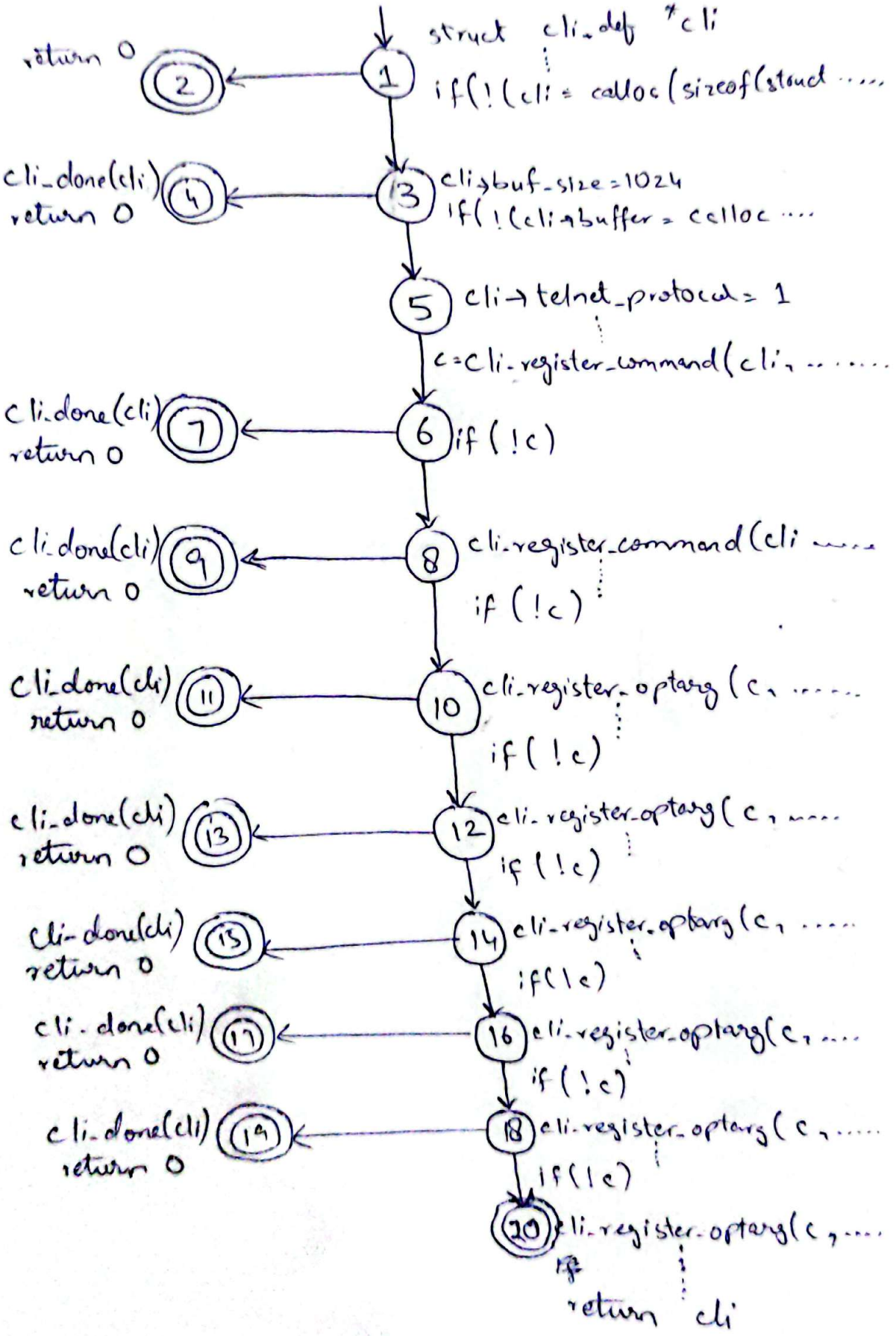
TR = {(1,2)(2,3),(3,4),(4,5),(5,3),(3,6),(6,7)(6,8)}

TP_1 = [1,2,3,4,5,3,6,7]

TP_2 = [1,2,3, ~~4,5,3~~ ,6,8]

Inputs: TP_1 TP_2

libcli.c


struct cli_def *cli
1
if(!(cli = calloc(sizeof(struct ....
return 0
2
3
cli->buf_size=1024
if(!(cli->buffer = calloc ....
cli_done(cli)
return 0
4
5
cli->telnet_protocol = 1
c=cli_register_command(cli, ...
6
if (!c)
cli_done(cli)
return 0
7
8
cli_register_command(cli ....
if (!c)
cli_done(cli)
return 0
9
10
cli_register_optarg (c, ...
if (!c)
cli_done(cli)
return 0
11
12
cli_register_optarg (c, ...
if (!c)
cli_done(cli)
return 0
13
14
cli_register_optarg(c, ...
if(!c)
cli_done(cli)
return 0
15
16
cli_register_optarg(c, ...
if (!c)
cli_done(cli)
return 0
17
18
cli_register_optarg(c, ...
if(!c)
cli_register_optarg(c, ...
19
20
return cli

$TR = \{(1,2), (1,3)(3,4), (3,5), (5,6), (6,7), (6,8), (8,9),$
$(8,10), (10,11), (10,12), (12,13), (12,14), (14,15), (14,16),$
$(16,17), (16,18), (18,19), (18,20)\}$

$TP_1 = [1, 2]$

$TP_2 = [1, 3, 4]$

$TP_3 = [1, 3, 5, 6, 7]$

$TP_4 = [1, 3, 5, 6, 8, 9]$

$TP_5 = [1, 3, 5, 6, 8, 10, 11]$

$TP_6 = [1, 3, 5, 6, 8, 10, 12, 13]$

$TP_7 = [1, 3, 5, 6, 8, 10, 12, 14, 15]$

$TP_8 = [1, 3, 5, 6, 8, 10, 12, 14, 16, 17]$

$TP_9 = [1, 3, 5, 6, 8, 10, 12, 14, 16, 18, 19]$

$TP_{10} = [1, 3, 5, 6, 8, 10, 12, 14, 16, 18, 20]$