TECHNICAL MEMO – PES Assignment 5 – Optimization

BASIC CODE FLOW ISHA Secure Algorithm

Reference - [RFC 8018 - PKCS #5: Password-Based Cryptography Specification Version 2.1 (ietf.org)]

STEPS to IMPLEMENT PBKDF2 as per RFC 8018 Section 5.2 is as follows:

Steps:

1. If dkLen > (2^32 - 1) * hLen, output "derived key too long"
   and stop.

2. Let l be the number of hLen-octet blocks in the derived key,
   rounding up, and let r be the number of octets in the last
   block:

   l = CEIL (dkLen / hLen)
   r = dkLen - (l - 1) * hLen

   Here, CEIL (x) is the "ceiling" function, i.e., the smallest
   integer greater than, or equal to, x.

3. For each block of the derived key apply the function F defined
   below to the password P, the salt S, the iteration count c,

and the block index to compute the block:

T_1 = F (P, S, c, 1) ,

T_2 = F (P, S, c, 2) ,

...

T_l = F (P, S, c, l) ,

where the function F is defined as the exclusive-or sum of the first c iterates of the underlying pseudorandom function PRF applied to the password P and the concatenation of the salt S and the block index i:

F (P, S, c, i) = U_1 \xor U_2 \xor ... \xor U_c

where

U_1 = PRF (P, S || INT (i)) ,

U_2 = PRF (P, U_1) ,

...

U_c = PRF (P, U_{c-1}) .

Here, INT (i) is a four-octet encoding of the integer i, most significant octet first.

4. Concatenate the blocks and extract the first dkLen octets to produce a derived key DK:

DK = T_1 || T_2 || ... || T_l<0..r-1>

5. Output the derived key DK.

time_pbkdf2_hmac_isha()

This function calls pbkdf2_hmac_isha() once and prints the run time duration in msec

When pbkdf2_hmac_isha() function is called – password, salt, lengths is passed for ISHA algorithm process

This returns success/failure of test cases and the time taken for the execution in msec

pbkdf2_hmac_isha()

This function implements HMAC-ISHA as defined in RFC 8018 section 5.2. We have defined ISHA_DIGESTLEN which is hLen – Length in octets

This function calls F() by passing password, password length, salt, salt length, iterations, block index to computer 20:8 bit blocks using "I"

After the execution of all the block, it concatenates the accumulator data that is derived key into DK

F()

This function is defined as the exclusive-or sum of the first c iterates of the underlying pseudorandom function PR applied to the password P and the concatenation of the salt S and the block index i

This function performs hashing and appends salt and password 4096 times

hmac_isha()

This function implements the HMAC-ISHA on key and message

It follows the steps mentioned to implement which is as follows –

- Padding the zero if key_len is greater than ISHA_BLOCKLEN
- If key_len less than ISHA_BLOCKLEN, copying the key into keypad and zero padding the result
- The task here is to bring it to 64 byte length after performing the appropriate function
- Further, ipad and opad are defined to perform the ISHA on the given text data
- To compute HMAC on given text – XOR operation is performed on ipad and opad

- Since key length is 20 bytes and block length is 64 bytes after performing XOR padding is done with zero
- XOR of ipad is done repeatedly with 0x36
- XOR of opad is done repeatedly with 0x5C
- Inner ISHA is performed by calling ISHAReset(), ISHAInput(), ISHAProcessMessageBlock() and ISHAResult()
- Outer ISHA is performed by calling ISHAReset(), ISHAInput(), ISHAProcessMessageBlock() and ISHAResult()

ISHAReset()

This function is used to restore an ISHA context to its default state. You can think of this as an initialization function.

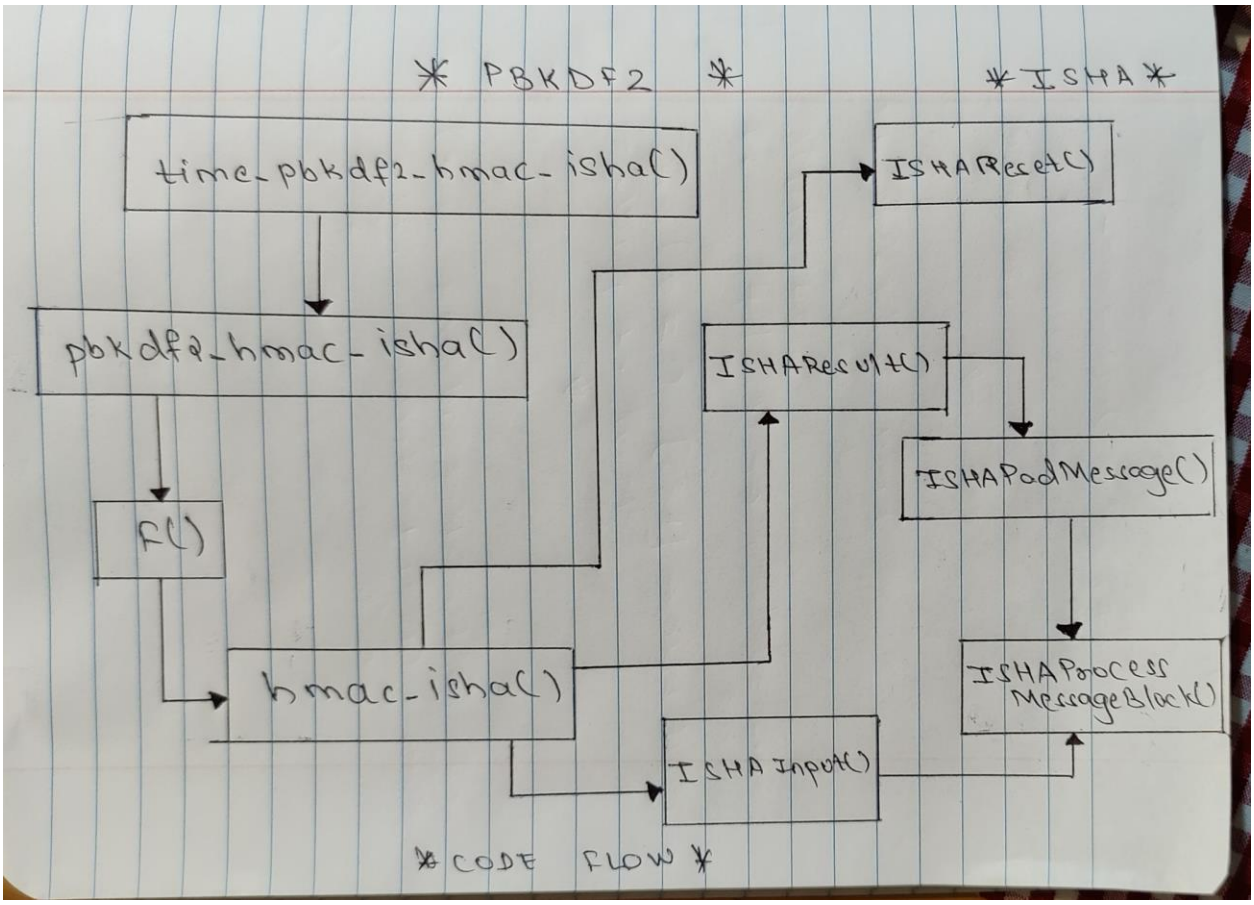
ISHAInput()

This function is used to push bytes into the ISHA hashing algorithm. After a call to ISHAReset, ISHAInput may be called as many times as needed Like SHA-1, the ISHA algorithm can hash up to 2^61 bytes.

ISHAResult()

This function is called once all bytes have been input into the algorithm. This function performs some padding and **final** computations, and then outputs the ISHA hash—a 160-bit (20 byte) value.

Flow Tree for better understanding



Profiling Analysis on Original code – Single time call time values

F() is called 3 times – 2914 X3 = 8742 msec

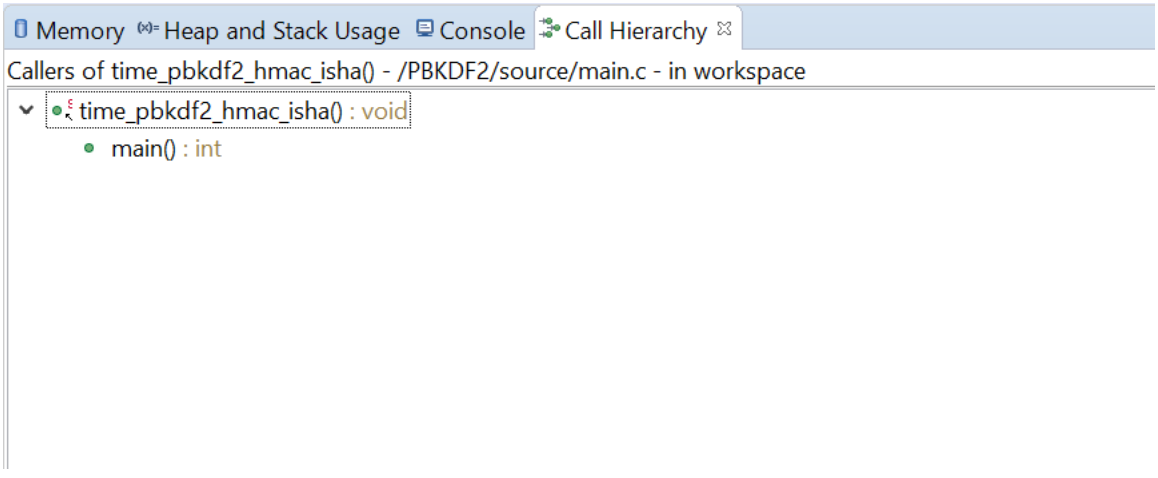
Function	Time Consumed in msec
F()	2914
hmac_isha()	0.7
ISHAReset()	0.024
ISHAInput()	0.163
ISHAResult()	0.013
ISHAProcessMessageBlock()	0.058
ISHAPadMessage()	0.085

Number of times each function is called as follows (Approximate Results)

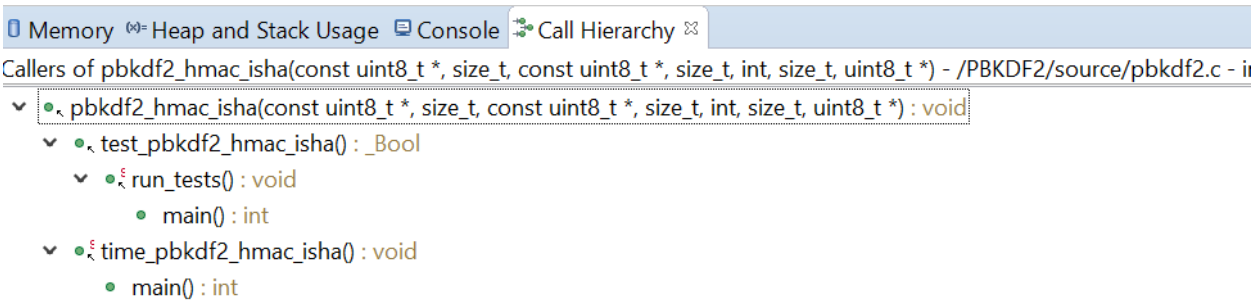
Functions	No of calls
pbkdf2_hmac_isha()	1
F()	3
hmac_isha()	4096
ISHAReset()	26668
ISHAInput()	53356
ISHAResult()	26668
ISHAProcessMessageBlock()	53356
ISHAPadMessage()	26668

CALL STACK ANALYSIS

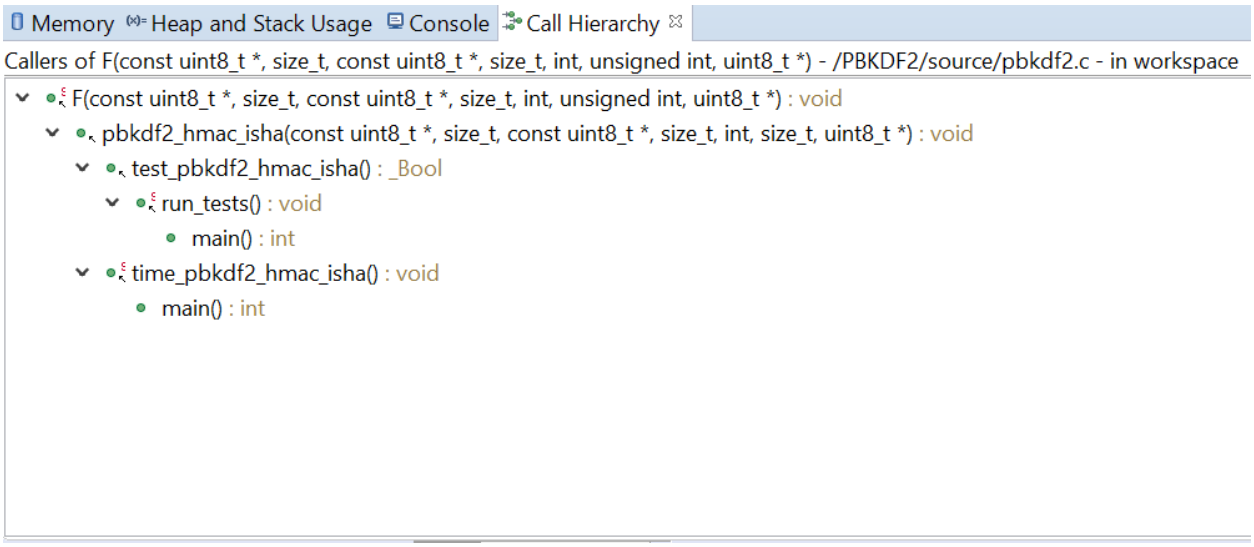
time_pbkdf2_hmac_isha()



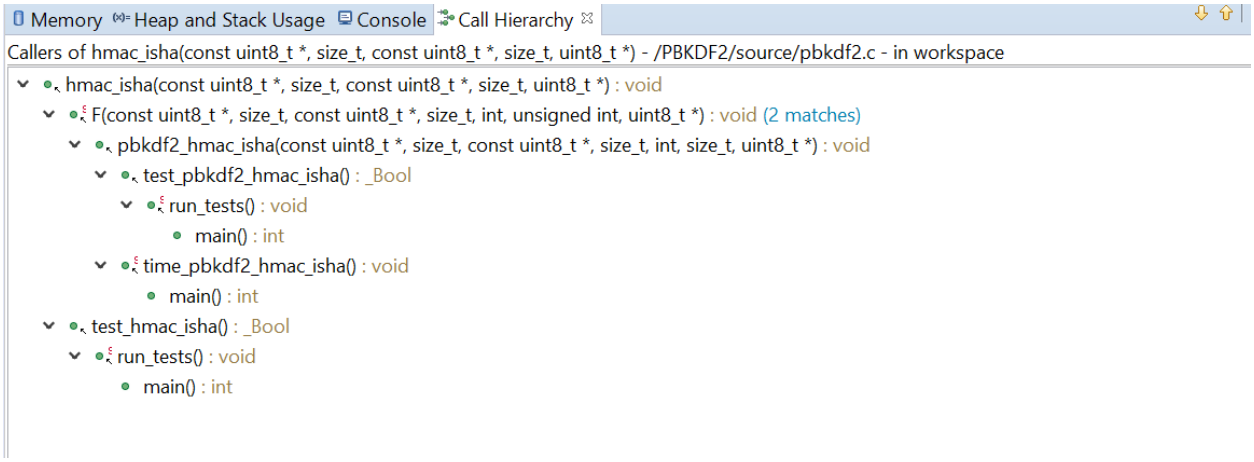
pbddf2_hmac_isha()


Memory
Heap and Stack Usage
Console
Call Hierarchy
Callers of pbkdf2_hmac_isha(const uint8_t *, size_t, const uint8_t *, size_t, int, size_t, uint8_t *) - /PBKDF2/source/pbkdf2.c - i
pbkdf2_hmac_isha(const uint8_t *, size_t, const uint8_t *, size_t, int, size_t, uint8_t *) : void
test_pbkdf2_hmac_isha() : _Bool
run_tests() : void
main() : int
time_pbkdf2_hmac_isha() : void
main() : int

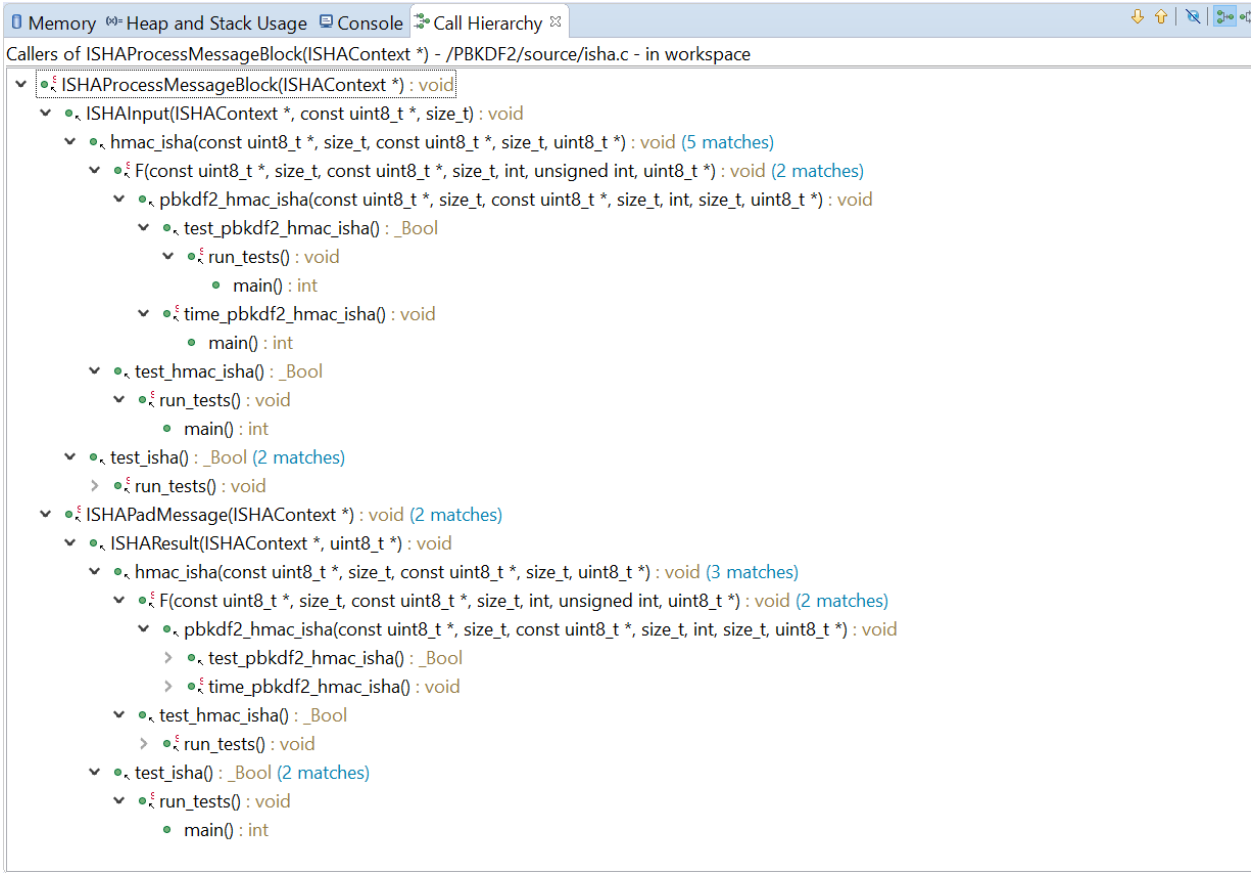
F()


Memory
Heap and Stack Usage
Console
Call Hierarchy
Callers of F(const uint8_t *, size_t, const uint8_t *, size_t, int, unsigned int, uint8_t *) - /PBKDF2/source/pbkdf2.c - in workspace
F(const uint8_t *, size_t, const uint8_t *, size_t, int, unsigned int, uint8_t *) : void
pbkdf2_hmac_isha(const uint8_t *, size_t, const uint8_t *, size_t, int, size_t, uint8_t *) : void
test_pbkdf2_hmac_isha() : _Bool
run_tests() : void
main() : int
time_pbkdf2_hmac_isha() : void
main() : int

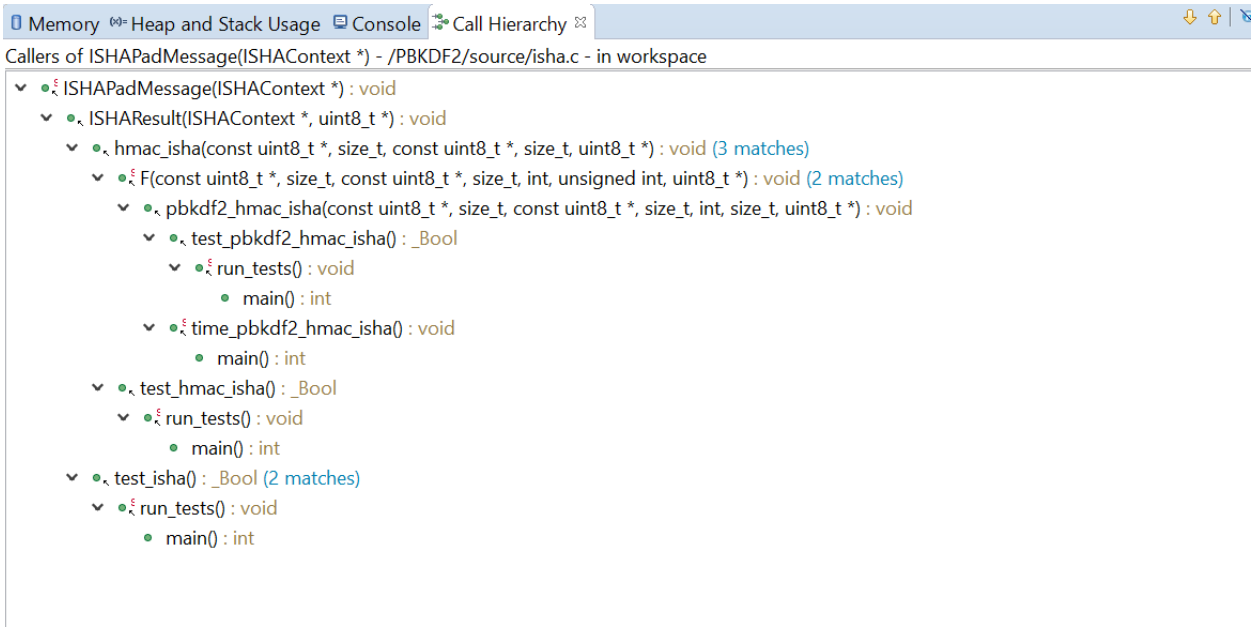
hmac_isha()


Memory
Heap and Stack Usage
Console
Call Hierarchy
Callers of hmac_isha(const uint8_t *, size_t, const uint8_t *, size_t, uint8_t *) - /PBKDF2/source/pbkdf2.c - in workspace
hmac_isha(const uint8_t *, size_t, const uint8_t *, size_t, uint8_t *) : void
F(const uint8_t *, size_t, const uint8_t *, size_t, int, unsigned int, uint8_t *) : void (2 matches)
pbkdf2_hmac_isha(const uint8_t *, size_t, const uint8_t *, size_t, int, size_t, uint8_t *) : void
test_pbkdf2_hmac_isha() : _Bool
run_tests() : void
main() : int
time_pbkdf2_hmac_isha() : void
main() : int
test_hmac_isha() : _Bool
run_tests() : void
main() : int

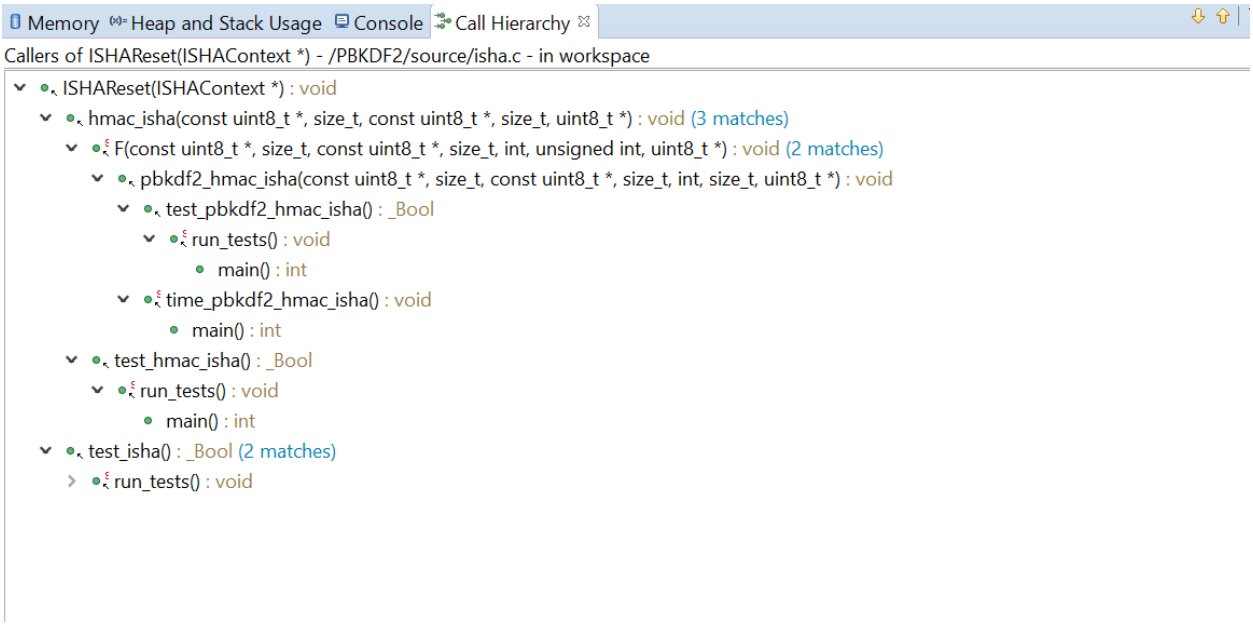
ISHAProcessMessageBlock()



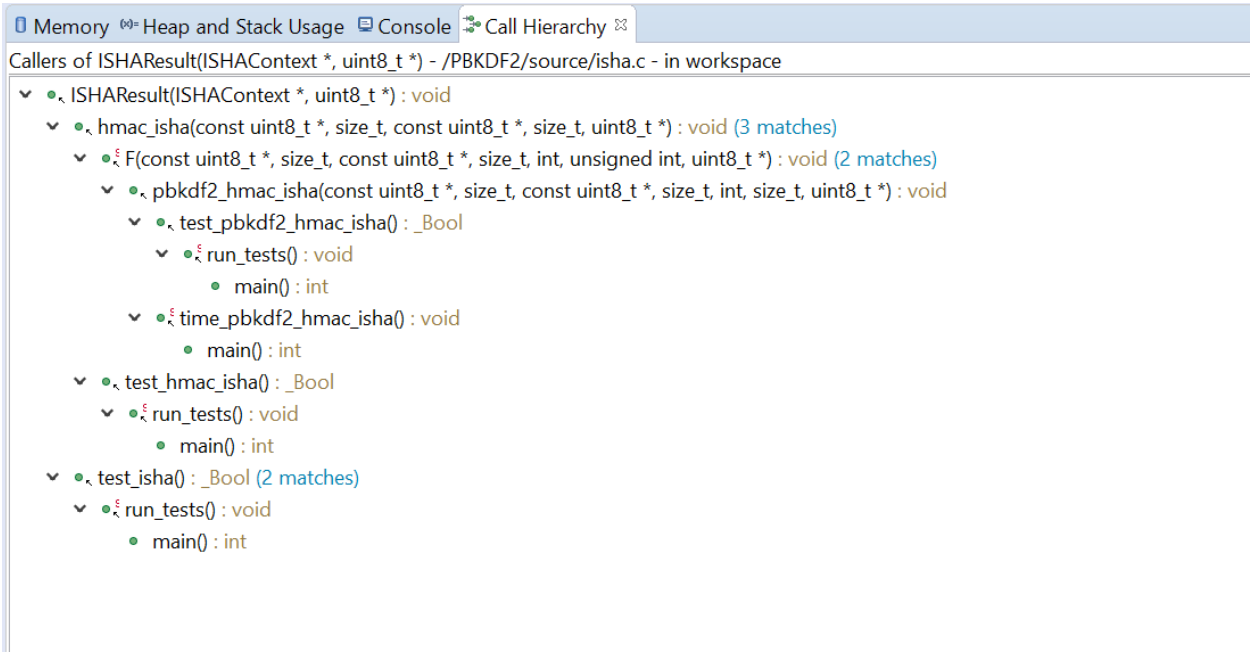
ISHAPadMessage()



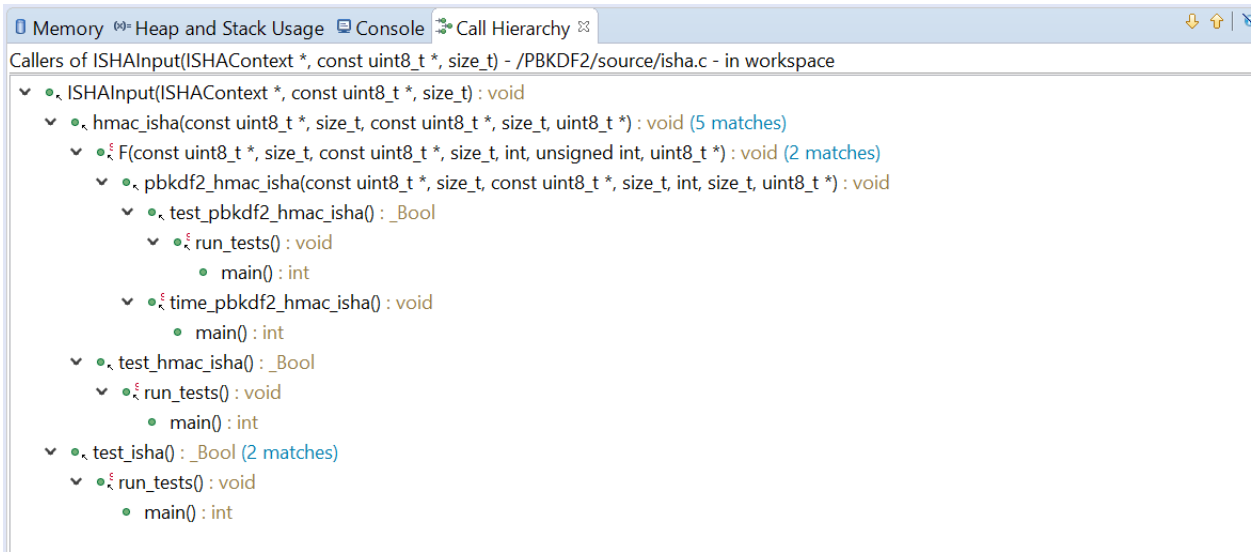
ISHAReset()



ISHAResult()



ISHAInput()



The function which have maximum number of calls from all other function should be looked into optimization at first. This is give a better approach by changing the code flow without changing its functionality and checking the updated run time.

SIZE .TEXT ANALYSIS

Original .text segment size

000005bc l F .text 00000154 time_pbkdf2_hmac_isha

000009b0 g F .text 00000130 pbkdf2_hmac_isha

000007d4 l F .text 000001dc F

00003eee g F .text 00000186 hmac_isha

0000055c g F .text 00000060 ISHAReset

00003d80 g F .text 000000c0 ISHAResult

00003e40 g F .text 000000ae ISHAInput

00003c72 l F .text 0000010e ISHAPadMessage

00003b20 l F .text 00000152 ISHAProcessMessageBlock

00000710 l F .text 00000078 run_tests

Total – 0x00000A8C

Optimized .text segment size

```
000005b8 l     F .text      00000154 time_pbkdf2_hmac_isha
00003e14 g     F .text      0000005a pbkdf2_hmac_isha
00003bd6 l     F .text      0000023e F
00003af0 g     F .text      000000e6 hmac_isha
0000055c g     F .text      0000005c ISHAReset
000038fe g     F .text      0000015a ISHAResult
00003a58 g     F .text      00000098 ISHAInput
0000381c l     F .text      000000bc ISHAProcessMessageBlock
0000070c l     F .text      00000078 run_tests
```

Total – 0x00000854

Difference of .text segment size = Original total – Optimized total

➔ 0x0A8C – 0x0854 = 0x0238 (568 in decimal)