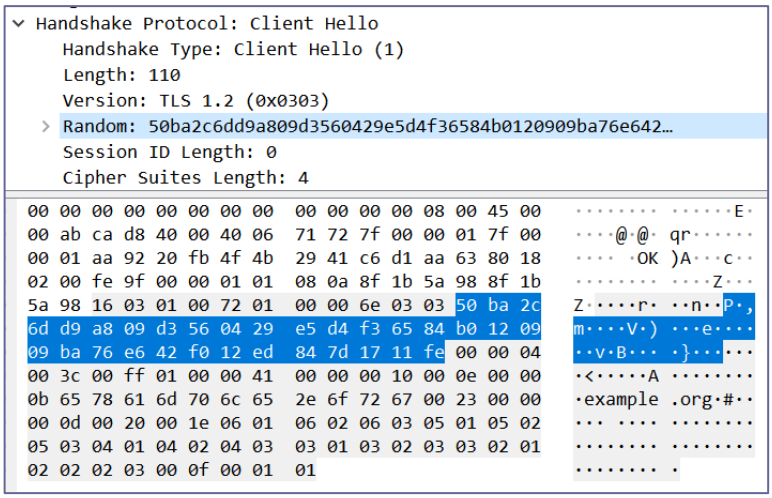
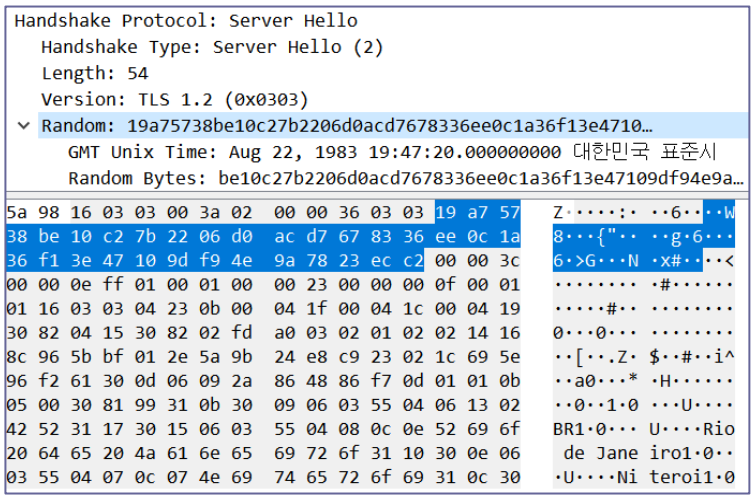
Lab2 (forward-secrecy) Report

2017313008 Kim Tae Eun

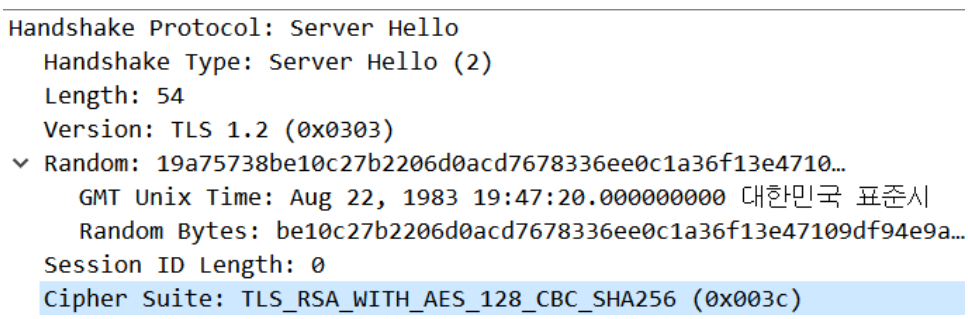
How to solve:

First of all, open traffic.pcap with wireshark and get the values of client_random and server_random.



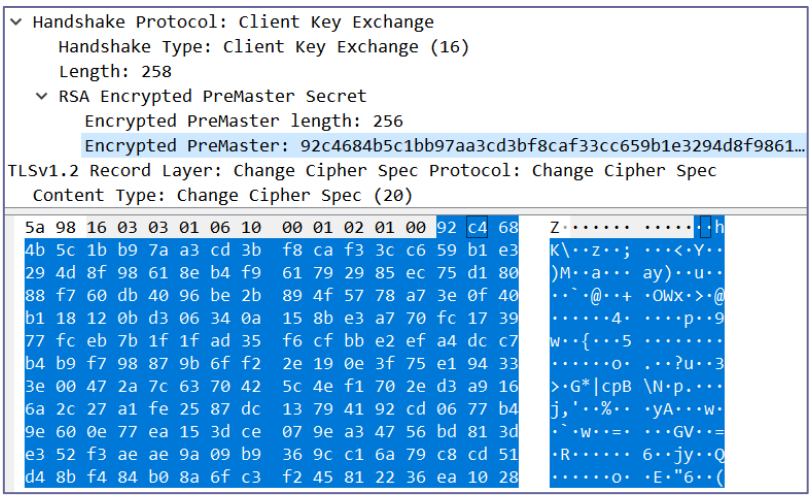



```
client_random = '50ba2c6dd9a809d3560429e5d4f36584b0120909ba76e642f012ed847d1711
fe'
server_random = '19a75738be10c27b2206d0acd7678336ee0c1a36f13e47109df94e9a7823ec
c2'
```



Cipher Suite: TLS_RSA_WITH_AES_128_CBC_SHA256

1. Decrypt the premaster secret and print it (10 pts)



Get the encrypted premaster secret value from wireshark.

```
"""
1. Decrypt the premaster secret and print it (10 pts)
"""


# Your code goes here
encrypted_premaster_secret = '92c4684b5c1bb97aa3cd3bf8caf33cc659b1e3294d8f98618
eb4f961792985ec75d18088f760db4096be2b894f5778a73e0f40b118120bd306340a158be3a770
fc173977fceb7b1f1fad35f6cfbbe2efa4dcc7b4b9f798879b6ff22e190e3f75e194333e00472a7
c6370425c4ef1702ed3a9166a2c27a1fe2587dc13794192cd0677b49e600e77ea153dce079ea347
56bd813de352f3aeae9a09b9369cc16a79c8cd51d48bf484b08a6fc3f245812236ea10285ce347e
41a93f0a398ec6f8b8b2edcd55d10fe35bb88ebbabb556d6d42544886f462bce76c1515b6ad0ed1
f547cf4a1a9ba423853ffa99d174dfba8071d6808155ab4d9ac6866a472df7a77106'


k=PrivKeyRSA('key.pem')
premaster_secret=k.decrypt(bytes(hex_to_data(encrypted_premaster_secret)))


# Your answer should be printed here
print('Decrypted Premaster Secret: %s\n' % data_to_hex(premaster_secret))
```

Decrypt encrypted_premaster_secret using the server's RSA private key (key.pem):

At this time, ‘decrypt’ function of PrivKeyRSA class is used.

And store the value in premaster_secret.

Result:

```
Decrypted Premaster Secret: 030314219f293c56e151d72cbb5b9284f9979632ac3684d0669
3163ae511137f3d02b306c819b4ca9f197cb1ae2ac200
```

2. Caclulate Master Secret (20 pts)

```
"""
2. Caclulate Master Secret (20 pts)
"""

# Your code goes here


f=PRF(PRF_ALGORITHM, TLS_12)
master_secret = f.compute_master_secret(premaster_secret, hex_to_data(client_ra
ndom), hex_to_data(server_random))


# Your answer should be printed here
print('Master Secret: %s\n' % data_to_hex(master_secret))
```

Calculate matser_secret value by using premaster_secret, client_random, and server_random values.

At this time, the ‘compute_master_secret’ function of the PRF class is used.

And this value is stored in master_secret. master_secret length is 48bytes.

Result:

```
Master Secret: 451681f7bc0e69c02a2c099413e595ee7f16cf3665052d303544069dde81f423
c2623278c2be4abdbed20cb0f8710b12
```

3. Calculate the Following (30 pts):

```
key_block=f.derive_key_block(master_secret, hex_to_data(server_random), hex_to_
data(client_random), 128)


# Your answer should be printed here
print('Key Block: %s \n' % data_to_hex(key_block))
```

Calculate key_block value by using master_secret, client_random, and server_random values. At this time, the 'derive_key_block' function of the PRF class is used.

And this value is stored in key_block. key_block length is 128bytes.

Result:

```
Key Block: 0dcd6899cce41f94b9f017b577d5eca6d5392a6d6ee14de9f24fbfc2b34650931d8a
4be9caaa860dc3f9eb110dabb323b2d98d908f7bbda133cdb102b1013320cccf743e338eff5e7c9
0598c73ac38eb13619556658b6a2c0c897c2cf40d7b8d53c4fc4e3f19695017d639d03bb5830fc9
8799823701b9f1529434c0793a7ade
```

```
"""
3. Calculate the Following (30 pts):


    1) Client Write Key,
    2) Client Write IV,
    3) Client Write Mac Key,
    4) Server Write Key,
    5) Server Write IV,
    6) Server Write Mac Key

Hint: KeyBlock Layout is as the following
- Byte000-Byte031: Client MAC Key
- Byte032-Byte063: Server MAC Key
- Byte064-Byte079: Client Write Key
- Byte080-Byte095: Server Write Key
- Byte096-Byte111: Client Write IV
- Byte112-Byte127: Server Write IV


"""
```

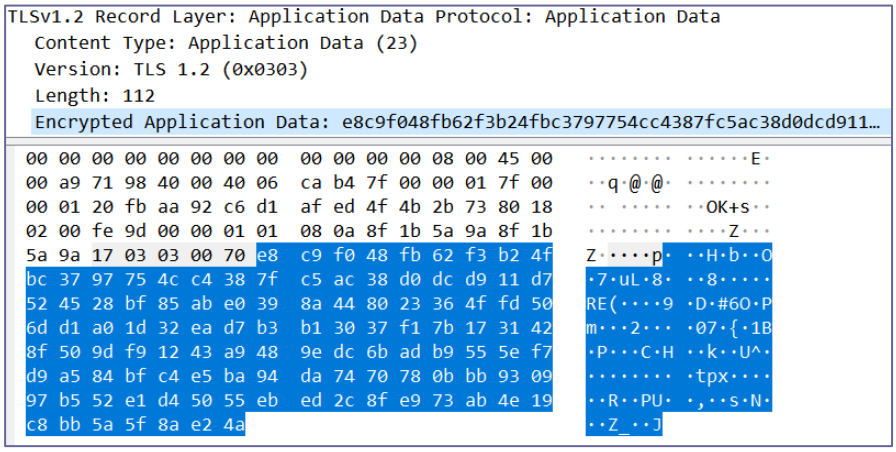
```
# Your code goes here

server_write_IV = data_to_hex(key_block[112:128])
client_write_IV = data_to_hex(key_block[96:112])
server_write_key = data_to_hex(key_block[80:96])
client_write_key = data_to_hex(key_block[64:80])
server_write_MAC_key = data_to_hex(key_block[32:64])
client_write_MAC_key = data_to_hex(key_block[0:32])



# Your answer should be printed here
print('Server Write IV: %s' % server_write_IV)
print('Client Write IV: %s' % client_write_IV)
print('Server Write Key: %s' % server_write_key)
print('Client Write Key: %s' % client_write_key)
print('Server Write MAC Key: %s' % server_write_MAC_key)
print('Client Write MAC Key: %s\n' % client_write_MAC_key)
```

Find the following symmetric keys used in AES according to the KeyBlock Layout.

Result:

```
Server Write IV: c98799823701b9f1529434c0793a7ade
Client Write IV: 53c4fc4e3f19695017d639d03bb5830f
Server Write Key: 13619556658b6a2c0c897c2cf40d7b8d
Client Write Key: cccf743e338eff5e7c90598c73ac38eb
Server Write MAC Key: 1d8a4be9caaa860dc3f9eb110dabb323b2d98d908f7bbda133cdb102b
1013320
Client Write MAC Key: 0dcd6899cce41f94b9f017b577d5eca6d5392a6d6ee14de9f24fbfc2b
3465093
```

4. Decrypt the Message, Get the Flag (40pts)



We should decrypt the message sent from server to client.

Store the encrypted application data sent from server port (8443) to client (43666) in encrypted_msg.

```
# Your code goes here

encrypted_msg=hex_to_data('e8c9f048fb62f3b24fbc3797754cc4387fc5ac38d0dcd911d7524528bf85abe0398a4480
23364ffd506dd1a01d32ead7b3b13037f17b1731428f509df91243a9489edc6badb9555ef7d9a584bfc4e5ba94da7470780
bbb930997b552e1d45055ebed2c8fe973ab4e19c8bb5a5f8ae24a')


aes=AES.new(bytes(hex_to_data(server_write_key)), AES.MODE_CBC, bytes(encrypted_msg[:16]))
flag=aes.decrypt(bytes(encrypted_msg[16:]))


# Your answer should be printed here
print('Flag: %s' % flag)
```

Plain text is encrypted by AES_128_CBC (symmetric), so the encrypted message can be decrypted using the symmetric key obtained above.

In the encrypted message, the previous 16 bytes are the IV values attached when encrypting.

Therefore, this value is used as the IV value, and the encrypted message can be decrypted using the server_write_key obtained above.

Result:

```
Flag: b'The flag is "n0wy0uund3rst4ndth31mp0rt4nc30ff0rw4rds3cr3cy"\x1f\x1e\x867\xe7`\xa4\x9fl\x85\
x84\x10,G\xf5O\x99nK3\xed\xa1\xed\x1e\xf1\x8d\x9fY\x98\x8c\xf0\xdb\x04\x04\x04\x04\x04'
```

The flag is “n0wy0uund3rst4ndth31mp0rt4nc30ff0rw4rds3cr3cy”

In the case of CBC Block Cipher, when encrypting, the MAC value for plaintext is first obtained and attached to plaintext, and then encryption is performed. Because it is encrypted in blocks of 16 bytes, if there are not enough bytes at the end, padding is used to fill the digits. Therefore, it can be seen from this result that 32 bytes after the flag are MAC values, and the next 5 bytes are padding.