PSL EX.2. (OFMC) Notes (updated: attack prevented!)

how is mac working in ofmc?
(if it is the other way around, (i.e. the first argument is the key that generates a mac of the subsequent argument(s) {this is how mac is represented in the ofmc documentation}) just change the argument..)

2.1.

1. A generates a nonce and sends it to B.
2. B generates a new nonce and appends it to the nonce sent from A. Subsequently, B uses the key derivation function to generate a new key by inputting the shared key and the nonces. Furthermore, with the nonces, B creates a mac for the output it gets from the key derivation function (another way around for mac would give the argument: B generates a mac for the nonces with the new key). This mac, and the nonce B generated, is sent to A. Likewise, A can compute the new key from the key derivation function by following the similar steps as B.
3. Moreover, A also generates a mac in the same manner as B and compares it to the one sent by B. If the mac's are similar to each other, A is able to authenticate B since to the get a match, B must have used the shared secret key to generated the mac.

wtf is A doing?
A -> B : the following mac.. (and A,B)
the mac includes:

- the new key
- "outer" (wtf is this?) another shared "secret" between A and B
- N1,N2
- mac of the new key? using N1,N2?

So even if the implementation of mac is other way around, the statement would be: using the new key, A generates a mac for "outer", N1, N2, and the mac previously sent by B. What is the point of having "outer"? How can B use it? Is it in the shared knowledge of both entities? Yes it is! So if an intruder impersonates A, how does it get "outer"?
Upon reception, B also generates a mac with input similar to A. Since outer and the key is only known to A and B, therefore, if the mac matches then B is able to authenticate A.
So in conclusion, the initial key is the one already shared by A and B. Nonces and the initial key are used to generate a new key and a mac is then created for the nonces using the new key. Upon reception, both entities can compute a mac using similar arguments known to both of them. Comparing the mac at each stage, therefore, proves the authenticity of either entity and it can also be assumed that the message itself has not been tampered. (Because a completely different mac would be generated for a different message).

2.2. [needs clarification from TA]
Why is A generating a second nonce in the next session? Why is the intruder sending the same nonce generated by A back to A? Isn't B supposed to do that? Moreover, the mac that B was supposed to send to A is now generated by A and sent to i? So having an intruder intercept the connection makes sense but why is A not following the protocol?

2.3.
So the nonce is not encrypted and it is vulnerable upon transmission on an insecure channel. Moreover, B doesn't know about the session in which the nonce was generated. A could include a reqID to keep track of the sessions and encrypt this along with the nonce using the shared secret key between A and B which is: exp(exp(g,secretk(A)),secretk(B)). Furthermore, OFMC verifies the prevention of attacks.

{**SOLVED**: The problem is that OFMC gives a parse error when encrypting the message using the shared key (exp(exp(g,secretk(A)),secretk(B)). Furthermore, there is a verification for one session when the said arguments are encrypted using the public key of A or B (exp(g,secretk(A)) or exp(g,secretk(B)))->(it says in the exercise that these keys and g are public).} Turns out I was missing a parenthesis for the shared key.