Problem 1. [50 points]

This problem and program Protocol below is based on section 11.2 of the text. The client program has a thread that repeatedly opens and closes a session. The server has a thread that always waits to receive a session request; for each received request, it starts a new thread to handle that request. The attacker can send and receive messages as A. The server program uses a map to store the ids of its threads. Conventions concerning maps is on the next page.

Protocol(A, B) {
    chan ← []; // channel
    hst ← []; // conn history
    K ← random(); // master key
    startSystem(Server(B,A,K)); // server B
    startSystem(Client(A,B,K)); // client A
    startSystem(Attacker()); // attacker
}

Client(A, B, K) { // atomicity points: 1
    t ← startThread(doSessions());
    return;
}

function doSessions() {
    while (true) {
        nA ← random();
        tx([A,B,1,nA,0]);
        1: msg ← rx([B,A,1,.,.]);
           if (msg[4] = enc(K,nA)) {
               nB ← msg[3];
               S ← enc(-K,nA+nB);
               hst.append([A,S]);
               tx([A,B,2,nA,enc(K,nB)]);
           }
    }
}

Attacker() {
    α; // everything attacker has read
    <send message [A,B,...]> <receive message [B,A,...]>
}

Server(B, A, K) { // atomicity points: 1, 2
    Map t ← [];
    t[0] ← startThread(listen());
    return;
}

function listen() {
    while (true) {
        1: msg ← rx([A,B,1,.,.]);
           nA ← msg[3];
           if (nA ≠ 0 and msg[4] = 0)
               t[nA] ← startThread(serveClient(nA));
    }
}

function serveClient(nA) {
    nB ← random();
    tx([B,A,1,nB,enc(K,nA)]);
    2: msg ← rx([A,B,2,nA,.,.]);
       if (msg[4] = enc(K,nB)) {
           S ← enc(-K,nA+nB);
           hst.append([B,S]);
       }
}

For each part below, answer yes or no. If yes, come up with an argument. If no, come up with a counter-example.

a. Does Inv $A_1$ hold, where
   $A_1 : \psi(K)$ // attacker does not learn K

b. Does Inv $A_2$ hold, where
   $A_2 : ((A,p) \in \text{hst}) \Rightarrow \psi(p)$ // attacker does not learn any session key of A

c. Does Inv $A_3$ hold, where
   $A_3 : ((i,j \in \text{hst.keys}) \text{ and } i \neq j \text{ and hst[i][0] = hst[j][0] = A})$
   $\Rightarrow \text{hst[i][1] \neq hst[j][1]}$ // A uses a session key only once

d. Does Inv $A_4$ hold, where
   $A_4 : (i > 0 \text{ and hst[i] = [B,p])} \Rightarrow \text{hst[i-1] = [A,p]}$ // attacker cannot connect to the server as A

e. Can the attacker learn K by dictionary attack, assuming that K is a weak key.
Problem 2. [50 points]

Repeat problem 1 after changing the protocol so that the server uses key $K+1$ to respond to the client’s message and the client uses key $K-1$ to respond to the server’s message.

That is, modify program Protocol so that:

- Client A expects message $[B, A, 1, nB, enc(K+1, nA)]$ (instead of $[B, A, 1, nB, enc(K, nA)]$) in response to its message $[A, B, 1, nA, 0]$.

Answer parts a–e for this modified program.

Conventions (applicable outside this homework also)

Maps:
Collections of 2-tuples where the first element is a key and the second is a value, the keys are distinct, and map entries can be indexed by the key.

For a map $x$:
- $x.size$: number of 2-tuples in the map.
- $x.keys$: sequence of its keys.
- $x.vals$: sequence of its values.
- $x[j]$, where $j$ is a key, refers to the value associated with $j$; i.e., $x$ has the tuple $[j, x[j]]$.
- $x.add(j, v)$: adds the tuple $[j, v]$ to $x$, replacing any prior $[j, .]$ tuple.
- $x[j] ← v$: same as $x.add(j, v)$.
- $x.remove(j)$: removes any $[j, .]$ tuple.

For example, if $x$ is the map $[[2, 5], [4, 7], ['A', 8], ['Z', 'SRVR']]$, then the following hold:
$x.size = 4; x.keys = [2, 4, 'A', 'Z']; x.vals = [5, 7, 8, 'SRVR']; x['Z'] = 'SRVR'$.

Referring to thread-specific quantities:
If, in a predicate or in an argument, you need to refer to a quantity specific to a thread in server B, you can do so by prefixing its name with the thread id.

For example, $B.t[nA].nB$ refers to the $nB$ value of the instance of function serveClient being executed by thread $B.t[nA]$. Thus the following may be a desired property of Protocol:

$Inv \ (exists(A.nB, B.t[A.nA].nB) \Rightarrow A.nB = B.t[A.nA].nB)$

Strong key assumption:
Unless otherwise mentioned, assume that keys are strong.

Predicates with non-program free variables:
For brevity, we may write a predicate containing variables that do not appear in the program being analyzed. When evaluating the predicate at a program state, treat these variables as universally quantified.

For example, predicate $A_2$ is

$(\exists A.p \in hst) \Rightarrow \psi(p)$

Variable $p$ is not defined in Protocol. So when evaluating $A_2$ at a program state, we treat $A_2$ as

$forall(p: (\exists A.p \in hst) \Rightarrow \psi(p))$