Note

- This homework is more like a take-home exam.
  Your solution should meet the requirements of exam 1 solution.
  It will be graded in the same way.
  It will have much higher weight than other homeworks.
- You cannot ask questions about how to proceed, whether you are on the right track, etc.
  You can "translate" such questions to exam 1 solution, and we will answer those.
- You can do this homework individually or with one partner. It's entirely up to you.
  Can't find a partner? Not happy with your partner? Want to leave your partner? Not my concern.
- Your solution should be neat and readable.

Problem 1 [30 points]

This program below is a variation of the Otway-Reese protocol. It has an attacker, kdc `Z`, client `A` and server `B`. The attacker can read-write the channel and get `A`'s old password (only when `A` is between sessions).

```
Protocol(Z, A, B) { // kdc, client, server
   chan ← [];
   hst ← [];            // connect history
   kAZ ← random();      // initial A-Z key
   kBZ ← random();      // initial B-Z key
   startSystem(Attacker());
   startSystem(Kdc(Z,A,B,kAZ,kBZ));
   startSystem(Client(A,Z,B,kAZ));
   startSystem(Server(B,Z,A,kBZ));
}
```

```
Attacker() {
   α; // initially has A, B, Z, all programs

   // functions executable by attacker
   function rChan {α ← chan;}     // read chan

   function wChan(x) {chan ← x;}  // write chan

   function getPwdA() {  // get A.key iff A.t at 1
      if (A.t at 1) {
         α.append(A.key);
         A.key ← Z.keyA ← random();
      }
   }
}
```

```
Kdc(Z, A, B, kAZ, kBZ) {  // atomicity points: 1
   keyA ← kAZ;
   keyB ← kBZ;
   t ← startThread(client());
   return;

   function kdc() {
      while (true) {
     1:  msg ← rx([B,Z,.]);
         x ← dec(keyB, msg[2]);
         if (x.size = 4 and x[0,1] = [A,B]) {
            nB ← x[2];
            y ← dec(keyA, x[3]);
            if (y.size = 3 and y[0,1] = [A,B]) {
               nA ← y[2];
               kAB ← random();
               rA ← enc(keyA, [nA,kAB]);
               rB ← enc(keyB, [nB,kAB]);
               tx([Z,B,rA,rB]);
            }
         }
      }
}
```

```
Client(A, Z, B, kAZ) {  // atomicity points: 1, 2
   key ← kAZ;
   t ← startThread(client());
   return;

   function client() {
      while (true) {
     1:  nL ← random();
         tx([A,B,1,enc(key,[A,B,nL])]);
     2:  msg ← rx([B,A,.]);
         x ← dec(key,msg[2]);
         if (x.size = 2) and x[0] = nL) {
            kAB ← x[1];
            hst.append([A,kAB]);
            tx([A,B,2,enc(kAB,'HELLO')]);
         }
      }
   }
}
```

```
Server(B, Z, A, kBZ) { // atomicity points: 1,2,3
   key ← kBZ;
   t ← startThread(server());
   return;

   function server() {
      while (true) {
     1:  msg ← rx([A,B,1,.]);
         nL ← random();
         tx([B,Z,enc(key,[A,B,nL,msg[3]])];
     2:  msg ← rx([Z,B,.,.]);
         x ← dec(key,msg[3]);
         if (x.size = 2 and x[0] = nL) {
            kAB ← x[1];
            hst.append([B,1,kAB]);
            tx([B,A,msg[2]]);
     3:     msg ← rx([A,B,2,.]);
            if (dec(kAB,msg[3]) = 'HELLO')
               hst.append([B,2,kAB]);
         }
      }
   }
}
```

Problem 1 (cont)

Part a.

Does *Inv* A_1 hold, where

A_1 : `((j in hst.keys) and j > 0 and hst[j] = [A,p])` $\Rightarrow$ `hst[j-1] = [B,1,p]`

If yes, assume that A appends [A,p] to hst at time t_0 and prove that [B,1,p] is the last entry in hst just before t_0.
If no, come up with a counter-example evolution, i.e., ending in a state where A_1 does not hold.

Part b.

Does *Inv* A_2 hold, where

A_2 : `((j in hst.keys) and j > 0 and hst[j] = [B,2,p])` $\Rightarrow$ `hst[j-1] = [A,p]`

If yes, assume that B appends [B,2,p] to hst at time t_0 and prove that [A,p] is the last entry in hst just before t_0.
If no, come up with a counter-example evolution, i.e., ending in a state where A_1 does not hold.

(**Hint:** *Inv* ψ(A.key) may hold. *Inv* ψ(B.key) may not hold.)

Problem 2 [30 points]

Repeat problem 1 after changing the kdc-to-server message to include the response to A inside the response to B. The change can be made as follows:

- In function kdc

```
...
rA ← enc(keyA, [nA,kAB]);
rB ← enc(keyB, [nB,kAB]);
tx([Z,B,rA,rB]);
...
```

becomes

```
...
rA ← enc(keyA, [nA,kAB]);
rB ← enc(keyB, [nB,kAB,rA]);
tx([Z,B,rB]);
...
```

- In function server

```
   ...
2: msg ← rx([Z,B,.,.]);
   x ← dec(key,msg[3]);
   if (x.size = 2 and x[0] = nL) {
   ...
   tx([B,A,msg[2]]);
   ...
```

becomes

```
   ...
2: msg ← rx([Z,B,.]);
   x ← dec(key,msg[2]);
   if (x.size = 3 and x[0] = nL) {
   ...
   tx([B,A,x[2]]);
   ...
```
