3 problems over 3 pages. 50 points. *Closed book. Closed notes. No calculator or electronic device.*

1. [20 points]

Here is secret-key authentication protocol between client A and server B.

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

```
Attacker() {
   α; // initially has A, B, all programs
   // functions executable by attacker
   function rChan {α ← chan;}    // read chan
   function wChan(x) {chan ← x;} // write chan
}
```

```
Client(A, B, K) {  // atomicity points: 1
   nL ← 0;
   t ← startThread(client());
   return;

   function client() {
      while (true) {
         nL ← nL + 2;
         tx([A,B,1,nL]);
     1:  msg ← rx([B,A,1,.,.]);
         if (msg[4] = enc(K,nL)) {
            nR ← msg[3];
            S ← enc(-K, nL+nR);
            hst.append([A,S]);
            tx([A,B,2,enc(K,nR)]);
         }
      }
   }
}
```

```
Server(B, A, K) { // atomicity points: 1,2
   nL ← 0;
   t ← startThread(server());
   return;

   function server() {
      while (true) {
     1:  msg ← rx([A,B,1,.]);
         nR ← msg[3];
         nL ← nL + 3;
         tx([B,A,1,nL,enc(K,nR)]);
     2:  msg ← rx([A,B,2,.]);
         if (msg[3] = enc(K,nL)) {
            S ← enc(-K, nL+nR);
            hst.append([B,S]);
         }
      }
   }
}
```

For each part, say whether its claim is true or false. If true, come up with an argument.
If false, come up with a counter-example evolution (in parts a,b) or a dictionary attack(in part c).

a. *Inv* A_1 holds, where
   $A_1 : \psi(\mathsf{K})$

b. *Inv* A_2 holds, where
   A_2 : ((i in hst.keys) and hst[i] = [B,p]) ⇒ ([A,p] in hst[0..i-1])

c. Attacker cannot learn K by dictionary attack, assuming that K is a weak key.

2. [10 points]

Here is a Diffie-Hellman protocol with public parameters g and p (known to all).

```
Protocol(A, B) {
   chan ← [];                    // channel
   K ← random();                 // A-B key
   startSystem(Attacker());
   startSystem(Client(A,B,K));
   startSystem(Server(B,A,K));
}
```

```
Attacker() {
   α; // initially has A, B, g, p, all programs
   // functions executable by attacker
   function rChan {α ← chan;} // read chan
   function wChan(x) {chan ← x;} // write chan
}
```

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

   function client() {
      nL ← random();
      tL ← g^nL mod p;
      tx([A,B,1,enc(K,tL)]);
   1: msg ← rx([B,A,1,.]);
      tR ← msg[3];
      keyDH ← tR^nL mod p;
      data ← random();
      tx([A,B,2,enc(keyDH,['HELLO',data])]);
   }
}
```

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

   function server() {
   1: msg ← rx([A,B,1,.]);
      tR ← dec(K,msg[3]);
      nL ← random();
      tL ← g^nL mod p;
      keyDH ← tR^nL mod p;
      tx([B,A,1,tL]);
   2: msg ← rx([A,B,2,.]);
      data ← dec(keyDH,msg[3])[1];
   }
}
```

Can the attacker obtain data by dictionary attack, assuming K is a weak key?

If you answer no, come up with an argument.
If you answer yes, come up with an evolution ending with the attacker obtaining data.

3. [20 points] Here is a Needham-Schroeder protocol with KDC Z, client A, server B, and attacker.

```
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));
}
```

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

   function client() {
      while (true) {
         n1 ← random();
         tx([A,Z,B,n1]);
    1:   msg ← rx([Z,A,.]);
         z ← dec(key,msg[2]);
         if (z.size = 4 and z[0,1] = [n1,B]) {
            kAB ← z[2];
            tkt ← z[3];
            n2 ← random();
            tx([A,B,tkt,enc(kAB,n2)]);
    2:      msg ← rx([B,A,.]);
            z ← dec(kAB,msg[2]);
            if (z.size = 2 and z[0] = n2-1) {
               n3 ← z[1];
               hst.append([A,kAB]);
               tx([A,B,enc(kAB,n3-1)]);
            }
         }
      }
   }
}
```

```
Kdc(Z, A, B, kAZ, kBZ) {
   // atomicity points: 1
   keyA ← kAZ;
   keyB ← kBZ;
   while (true) {
  1:  msg ← rx([A,Z,B,.]);
      kAB ← random();
      tkt ← enc(keyB,[kAB,A]);
      tx([Z,A,enc(keyA,[msg[3],B,kAB,tkt])]);
   }
}
```

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

   function server() {
      while (true) {
    1:   msg ← rx([A,B,.,.]);
         tkt ← msg[2];
         z ← dec(key,tkt);
         if (z.size = 2 and z[1] = A) {
            kAB ← z[0];
            n2 ← dec(kAB,msg[3]);
            n3 ← random();
            tx([B,A,enc(kAB,[n2-1,n3])]);
    2:      msg ← rx([A,B,.]);
            if (msg[2] = enc(kAB,n3-1))
               hst.append([B,kAB]);
         }
      }
   }
}
```

```
Attacker() {
   α; // everything attacker has read

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

   function writeChan(x) {chan ← x;}

   function getPwdA() {
      α.append(A.key);
      A.key ← Z.keyA ← random();
   }
}
```

For each part, say whether its claim is true or false. If true, come up with an argument.
If false, come up with a counter-example evolution.

a. *Inv* A_1 holds, where
   A_1 : ψ(A.mKey) // attacker does not learn A's current master key

b. *Inv* A_2 holds, where
   A_2 : ((j in hst.keys) and j > 0 and hst[j] = [B,p]) $\Rightarrow$ hst[j−1] = [A,p]