3 problems over 3 pages. 60 points. Closed book. Closed notes. No calculator or electronic device.

1. [20 points]

```
Protocol(A, B) {
   chan ← [];
   hst ← [];
   mKey ← random();
   startSystem(A, Client(A,B,mKey));
   startSystem(B, Server(B,A,mKey));
   startSystem(Attacker());
}
```

```
Attacker() {
   read and write chan
}
```

```
Client(A, B, mKey) {
   // atomicity points: 1
   nL ← 0;
   while (true) {
      nL ← nL + 5;
      tx([A,B,nL]);
 1:   msg ← rx([B,A,.,.]);
      if (msg[3] = enc(mKey,nL)) {
         nR ← msg[2];
         hst.append([A,nL,nR]);
         tx([A,B,nL,enc(mKey,nR)]);
      }
   }
}
```

```
Server(B, A, mKey) {
   // atomicity points: 1,2
   nL ← 0;
   while (true) {
 1:   msg ← rx([A,B,.]);
      nR ← msg[2];
      nL ← nL + 7;
      tx([B,A,nL,enc(mKey,nR)]);
 2:   msg ← rx([A,B,.,.]);
      if (msg[2] = nR and msg[3] = enc(mkey,nL)) {
         hst.append([B,nL,nR]);
      }
   }
}
```

For each assertion below, prove or disprove whether the assertion holds for Protocol. If you prove, present an invariantly-complete predicate that implies the assertion's predicate. If you disprove, present a counter-example evolution.

a. *Inv* (mKey *ncf* α)

b. *Inv* forall(i in hst.keys: hst[i] = [B,nB,nA] $\Rightarrow$ [A,nA,nB] in hst[0..i-1])

Solution to part a [10 points]

Informal argument [4 points]

It suffices to show that any `mKey`-term in α is a secure encryption using `mKey` (because of axiom 2). The messages sent by `A` or `B` contain three kinds of fields: ids (`A`, `B`), `nL` values, and `enc(mKey,nR)` values. Only the last kind involve `mKey`, and these are secure encryptions using `mKey` because even though the attacker can write to `chan`, the `mKey` values it can write are themselves secure encryptions using `mKey`.

Proof [6 or 10 (if Informal argument absent)]

The conjunction of C_1–C_4 is invariantly complete, and C_1 implies (`mKey` *ncf* α).

C_1 : (`y in` α.*inpts*(`mKey`)) $\Rightarrow$ (`y` *seu* `mKey`) [2 points]

C_2 : (`y in chan`.*inpts*(`mKey`)) $\Rightarrow$ (`y` *seu* `mKey`) [2 points]

C_3 : `A.nL`.*inpts*(`mKey`) = [] [1 point]

C_4 : `B.nL`.*inpts*(`mKey`) = [] [1 point]

Details:

	C_1	C_2	C_3	C_4
initial step	true	true	true	true
A.1	C_2, C_3, C_1	C_2, C_3	true	C_4
B.1	C_2, C_4, C_1	C_2, C_4	C_3	$true$
B.2	C_1	C_2	C_3	C_4
attacker write	C_1	C_1, C_2	C_3	C_4

Solution to part b [10 points]

We disprove the assertion.

Informal argument [4 points]

Before an execution of `B.1`, The attacker knows the `B.1` When `B` is at `1`, the attacker knows the challenge, say `nB`, that `B.1` will issue next. So it writes message `[A,B,1,nB]` to `chan`. `B` responds with message `[B,A,nB,enc(mKey,nB)]`. The attacker, using the last field of this message, makes up the response expected by `B.2`, which causes `B` to add an entry to `hst` without `A` receiving any message.

Proof [6 points]

Counter-example evolution:

- Initial step
  After: `[A,B,1]` in `chan`; `hst = []`.
- Attacker changes field 2 in the above message to `7`. [2 points]
- `B.1` [1 points]
  After: `[B,A,7,enc(mKey,7)]` in `chan`; `B.nL = B.nR = 7`; `hst = []`.
- Attacker, using `enc(mKey,7)` field in above message, sets `chan` to `[[B,A,enc(mKey,7)]]`. [2 points]
- `B.2` [1 points]
  After: `hst = [[B,7,7]]`. Assertion's predicate not satisfied.

–6 if attacker does not come up with `enc(mKey,B.nL)`

2. [20 points]

An organization has a PKI (public-key infrastructure) for its users consisting of a single certification authority (CA) and a single directory server (DS) that serves certificates and CRLs to any user. Certificates have an expiry time of 1 year. CRLs are issued weekly. Answer the following questions. Be brief and precise.

a. What is the minimum information that a user in the organization must remember. Is any of it secret.
b. What is the minimum information a certificate must have.
c. Describe the steps that a user A takes to establish a connection to a user B with a shared session key.

Solution to part a [3 points]

A user must remember at the minimum
– its own private key [1 point]
– the CA's public key [1 point]
Its private key is secret, shared with no one. [1 point]

Solution to part b [7 points]

A certicate has the following at a minimum:
– serial number [1 point]
– user id [1 point]
– user public key [2 point]
– expiry date [1 point]
– issuer (CA) id
– issue date
– signature on above by CA [2 point]

Solution to part b [10 points]

- `A` asks DS for `B`'s certificate, say `certB`, and latest CRL, say `CRL` (and `A`'s certificate if needed). [2 points]
- `A` verifies certificate(s) using the CA's public key. `A` verifies CRL using the CA's public key, checks `certB` not in `CRL` [1 points]
- `A` generates a session key, say `k`, signs it (with its private key), encrypts it (with `B`'s public key. [3 points]
- `A` sends `[A,B,enc(pubB,[k,enc(priA,k)]),certA,CRL]`.
- `B` gets and verifies `certA` and `CRL` (just as `A` did with `certB` and `CRL`) [3 points]
- `B` decrypts `enc(pubB,k)` to get `k`. [1 points]

3. [20 points]

Function enc(.,.) can encrypt arbitrary values, including integers (e.g., enc(K,3)) and structures (e.g., enc(K,[3,5])). Note that enc(K,3) is different from enc(K,[3]).

In the program below, the following happens repeatedly if the attacker does nothing:

- A sends [A,B,1,nA]
- B responds with [B,A,enc(mKey,[nB,nA+1])]
- A responds with [A,B,enc(mKey,[nA,nB+1])]

```
Protocol(A, B) {
   chan ← [];
   hst ← [];   // connection history
   mKey ← random();
   startSystem(A, Client(A,B,mKey));
   startSystem(B, Server(B,A,mKey));
   startSystem(Attacker());
}
```

```
Attacker() {
   read and write chan
}
```

```
Client(A, B, mKey) {
   // atomicity points: 1
   sKey ← random();
   while (true) {
      nL ← random();
      tx([A,B,1,nL]);
  1:  msg ← rx([B,A,.]);
      z ← dec(mKey,msg[2]);
      if (z.size = 2 and z[1] = nL+1) {
         nR ← z[0];
         hst.append([A,nL,nR]);
         sKey ← enc(mKey,nL+nR);
         tx([A,B,2,enc(mKey,[nL,nR+1])]);
      }
   }
}
```

```
Server(B, A, mKey) {
   // atomicity points: 1,2
   sKey ← random();
   while (true) {
  1:  msg ← rx([A,B,1,.]);
      nR ← msg[3];
      nL ← random();
      tx([B,A,enc(mKey,[nL,nR+1])]);
  2:  msg ← rx([A,B,2,.]);
      z ← dec(mKey,msg[3]);
      if (z.size = 2 and z[0] = nR and z[1] = nL+1) {
         hst.append([B,nL,nR]);
         sKey ← enc(mKey,nL+nR);
      }
   }
}
```

For each assertion below, prove or disprove whether the assertion holds for Protocol. If you prove, present an invariantly-complete predicate that implies the assertion's predicate. If you disprove, present a counter-example evolution.

a. *Inv* (mKey *ncf* α)

b. *Inv* (A.sKey *ncf* α)

c. *Inv* forall(i in hst.keys: hst[i] = [B,nB,nA] $\Rightarrow$ [A,nA,nB] in hst[0..i-1])

Solution to part a [3 points]

The conjunction of the following predicates is invariantly complete and implies `mKey` *ncf* α.

C_1: `(y in` α`.`*inpts*`(mKey))` $\Rightarrow$ `(y` *seu* `mKey)` [1 points]

C_2: `(y in chan.`*inpts*`(mKey))` $\Rightarrow$ `(y` *seu* `mKey)` [1 points]

C_3: `A.nL.`*inpts* `= []` [1 points]

Solution to part b [8 points]

We prove it.

Informal argument [4 points] Suppose `A.1` sets `A.sKey` to `enc(mKey,nA+nB)`. The attacker has `nA`, because it was previously sent in the open in a `[A,B,1,.]` message. But `nB` has not been sent in the open. So `A.sKey` is not computable by the attacker.

Proof [5 or 8 (if no informal argument) points] The conjunction of the following predicates is invariantly complete and implies `A.sKey` *ncf* α (assuming *Inv* `(mKey` *ncf* α`)`.

C_1: `(y in` α`.`*inpts*`(mKey))` $\Rightarrow$ `(y` *seu* `mKey) and y = enc(mKey,[int,int])` [2 points]

C_2: `(y in chan.`*inpts*`(mKey))` $\Rightarrow$ `(y` *seu* `mKey) and y = enc(mKey,[int,int])` [2 points]

C_3: `A.nL.`*inpts* `= []` [1 points]

Can also replace `y = enc(mKey,[int,int])` with `y is not enc(mKey,int)`. Without saying this, you get close to zero points. Saying `y = enc(mKey,nA+nB)` is no good.

Solution to part c [9 points]

We prove it.

Informal argument [4 points] Suppose `[B,nB,nA]` is added to `hst` at time t_0. Then just before t_0, `B.2` received message `[A,B,2,enc(mKey,[nA,nB+1])]` and `B.nL = nB`, `B.nR = nA`, and `B at 2` held. Let `B.nL` be set to `nB` at time t_1. So `B` does nothing between t_1 and t_0. The attacker does not have `mKey`. So message `[A,B,2,enc(mKey,[nA,nB+1])]` was sent by `A.1` at some time t_2 between t_1 and t_0. The same execution of `A.1` also entered `[A,nA,nB]` into `hst`.

Proof [5 points] The conjunction of the following predicates is invariantly complete and implies `A.sKey` *ncf* α (assuming *Inv* `(mKey` *ncf* α`)`.

E_0: `((i in hst.keys) and hst[i] = [B,nB,nA])` $\Rightarrow$ `([A,nA,nB] in hst[0..i-1])`

E_1: `((B at 2) and B.nL = nB and B.nR = nA and ([A,B,enc(mKey,[nA,nB+1])] in chan))` $\Rightarrow$ `([A,nA,nB] in hst)`

E_2: `((B at 2) and B.nL = nB and B.nR = nA and (enc(mKey,[nA,nB+1]) in` α`))` $\Rightarrow$ `([A,nA,nB] in hst)`