Assignment Report

By Shifat Ur Rahman

Model:

For Local attestation, I implemented a very simple model. There are two entities – Alice and Bob. On receiving request from Alice, Bob generates a fresh key **k** and outputs **aenc(sign((pkX, pkB, k), skB), pkX)** in channel **c**; that is, Bob's signature on the key **k** paired with his identity **spk(skB)** and encrypted for his client using her (Alice) public key **pkX**. Meanwhile, Alice waits for the response from Bob and upon receiving, verifies the ciphertext using the destructor **adec**. The statement **let (=pkA, =pkB, k: key) = checksign(sigB, pkB)** uses destructors and pattern matching with type checking to verify that **sigB** is a signature under **skB** containing a pair, where the first element is the server's public signing key and the second is a symmetric key **k**. If **sigB** is not the correct signature, then Alice will halt because no else condition is written here.

Summary of Results:

1. **Query not attacker(s[]) is true.**

The attacker was not able to obtain the free name s.

2. **Query event(aliceReceiveKey(x_2)) ==> event(bobFreshKey(x_2)) is true.**

The attacker was not able to obtain the information transferred during Bob generating fresh key, encrypting with Alice's public key and Alice receiving it.

3. **Query inj-event(aliceTerm(skx,pky)) ==> inj-event(bobAccepts(skx,pky)) is false.**

Alice is the one initiating the communication in the start. This is vulnerable to replay attacks, that is why it is returning false. We have to add a nonce to make this attack-proof.

4. **Query inj-event(bobTerm(skx)) ==> inj-event(aliceAccepts(skx)) is true.**

Authentication of BobTerm to AliceAceept event holds.