Christopher Hsu
CS259
Project Summary

In this project I studied the *i*KP protocol family. In particular, I examined the 1KP protocol in detail, and compared results drawn to documented results of 2KP and 3KP.

I modeled 1KP using Murφ to examine effects of removing components. Since the 1KP protocol deliberately ignores encryption (except that of credit card numbers), it was usually irrelevant to examine the knowledge that an eavesdropper could obtain. However, although there was no explicit encryption of messages sent on the wire, there are cryptographic primitives embedded into the individual components of those messages. The most challenging aspect in attempting to create an accurate Murφ description of 1KP was to see which cryptographic functions could be or needed to be abstracted.

As explained in the presentation, the SALT component is essential to prevent an eavesdropper listening on the Authentication-Request line (i.e. merchant to acquirer) from making a dictionary attack to determine the value of DESC (a description of the purchase order). Since the dictionary attack involves computing the hashed value of DESC (either by itself or concatenated with SALT), this was difficult to directly model with Murφ. Instead, I wrote a slightly different invariant to model a similar effect. The invariant states that no intruder ever knows both the SALT and DESC of the same customer. A failure of this invariant indicates that an eavesdropper can perform a dictionary attack, since he knows both DESC and SALT and can compute the hash value of their concatenation.

In effect, what I modeled was an alternate aspect of 1KP. Namely, instead of taking SALT out of the protocol, I modeled the ability of the intruder to learn SALT based on his power. This power is configurable to either allow the eavesdropper to listen on the Initiate line (i.e. customer to merchant) or not, based on the constant EVE_INIT. Learning SALT (and therefore violating the invariant) corresponds to the possibility of a dictionary attack, whereas when the invariant is satisfied, no dictionary attack is possible. As expected, when EVE_INIT is true (i.e. the eavesdropper listens on two separate lines), the invariant is violated. When EVE_INIT is false, the invariant is satisfied. Source code for the intruder behavior and the invariant, as well as a violating trace, is given at the end of this document.

I attempted to prove some documented unsatisfied requirements of 1KP, for example proof of authentication of merchant by customer. However, I found that Murφ seemed inappropriate for modeling such "proof" or "receipt" conditions. It was difficult to find a way to model knowledge of an agent, or equivalently, to model an agent's assurance of some aspect that is not explicitly included in the messages. For such properties a paper proof seems more appropriate.

```
// Intruder behavior, depends on EVE_INIT
-- intruder i intercepts messages
ruleset i: IntruderId do
  choose j: net do
    rule "Interception"
    !ismember (net[j].source, IntruderId)
  ==>
  var
    temp: Message;
    begin
      alias msg: net[j] do
        if msg.mType=M_AuthRequest then
          int[i].descs[msg.desc] := true;
        else
          if msg.mType=M_Initiate & EVE_INIT then
            int[i].salts[msg.salt] := true;
          end;
        end;
      end;
    end;
  end;
end;
```

```
// Invariant
-- Intruder never knows desc and salt from same customer
invariant "Intruder does not know DESC"
  forall i: IntruderId do
    forall j: CustomerId do
    !int[i].descs[j] |
    !int[i].salts[j]
  end
end;
```

```
// Trace for weak intruder (EVE_INIT = false):
Status:

        No error found.

State Space Explored:

        6 states, 9 rules fired in 0.30s.
```

```
// Trace for strong intruder (EVE_INIT = true):
Startstate Startstate 0 fired.
cus[CustomerId_1].state:C_SLEEP
cus[CustomerId_1].merchant:CustomerId_1
mer[MerchantId_1].state:M_SLEEP
mer[MerchantId_1].customer:MerchantId_1
mer[MerchantId_1].acquirer:AcquirerId_1
acq[AcquirerId_1].state:A_SLEEP
acq[AcquirerId_1].merchant:Undefined
int[IntruderId_1].salts[CustomerId_1]:false
int[IntruderId_1].salts[MerchantId_1]:false
int[IntruderId_1].salts[AcquirerId_1]:false
int[IntruderId_1].salts[IntruderId_1]:false
int[IntruderId_1].descs[CustomerId_1]:false
int[IntruderId_1].descs[MerchantId_1]:false
int[IntruderId_1].descs[AcquirerId_1]:false
int[IntruderId_1].descs[IntruderId_1]:false
----------

Rule Initiate, j:MerchantId_1, i:CustomerId_1 fired.
net{0}.source:CustomerId_1
net{0}.dest:MerchantId_1
net{0}.key:Undefined
net{0}.mType:M_Initiate
net{0}.salt:CustomerId_1
net{0}.cid:CustomerId_1
net{0}.mid:Undefined
net{0}.nonce:Undefined
net{0}.desc:Undefined
net{0}.encslip:Undefined
net{0}.yn:Undefined
```

```
net{0}.sig:Undefined
cus[CustomerId_1].state:C_WAIT
cus[CustomerId_1].merchant:MerchantId_1
----------

Rule Interception, i:IntruderId_1, j:0 fired.
int[IntruderId_1].salts[CustomerId_1]:true
----------

Rule Invoice, j:0, i:MerchantId_1 fired.
net{0}.source:MerchantId_1
net{0}.dest:CustomerId_1
net{0}.mType:M_Invoice
net{0}.salt:Undefined
net{0}.cid:Undefined
net{0}.mid:MerchantId_1
net{0}.nonce:MerchantId_1
mer[MerchantId_1].state:M_WAIT
mer[MerchantId_1].customer:CustomerId_1
----------

Rule Payment, j:0, i:CustomerId_1 fired.
net{0}.source:CustomerId_1
net{0}.dest:MerchantId_1
net{0}.mType:M_Payment
net{0}.mid:Undefined
net{0}.nonce:Undefined
net{0}.encslip:CustomerId_1
cus[CustomerId_1].state:C_AUTH
----------

Rule AuthRequest, j:0, i:MerchantId_1 fired.
net{0}.source:MerchantId_1
net{0}.dest:AcquirerId_1
net{0}.mType:M_AuthRequest
net{0}.salt:CustomerId_1
net{0}.desc:CustomerId_1
mer[MerchantId_1].state:M_AUTH
----------

Rule Interception, i:IntruderId_1, j:0 fired.
The last state of the trace (in full) is:
net{0}.source:MerchantId_1
net{0}.dest:AcquirerId_1
net{0}.key:Undefined
net{0}.mType:M_AuthRequest
net{0}.salt:CustomerId_1
net{0}.cid:Undefined
net{0}.mid:Undefined
net{0}.nonce:Undefined
net{0}.desc:CustomerId_1
net{0}.encslip:CustomerId_1
net{0}.yn:Undefined
net{0}.sig:Undefined
cus[CustomerId_1].state:C_AUTH
cus[CustomerId_1].merchant:MerchantId_1
mer[MerchantId_1].state:M_AUTH
mer[MerchantId_1].customer:CustomerId_1
mer[MerchantId_1].acquirer:AcquirerId_1
acq[AcquirerId_1].state:A_SLEEP
acq[AcquirerId_1].merchant:Undefined
int[IntruderId_1].salts[CustomerId_1]:true
int[IntruderId_1].salts[MerchantId_1]:false
int[IntruderId_1].salts[AcquirerId_1]:false
int[IntruderId_1].salts[IntruderId_1]:false
int[IntruderId_1].descs[CustomerId_1]:true
int[IntruderId_1].descs[MerchantId_1]:false
int[IntruderId_1].descs[AcquirerId_1]:false
int[IntruderId_1].descs[IntruderId_1]:false
----------
```

```
End of the error trace.

==========================================================================

Result:

        Invariant "Intruder does not know DESC" failed.

State Space Explored:

        11 states, 15 rules fired in 0.11s.
```