Exam 3 Report

Alfred Murabito

March 19, 2020

Abstract

This report answers the three questions in the CIS 634 Exam 3. The first question is answered in the code in the HOL/exam1Script.sml source file. The steps involving the shared-key encyrption scheme where a new key is generated for every message is answered in question 2. Question 3 involves what needs to be done to facilitate using HOL in artiifical intelligence.

Acknowledgments: I received no assistance with this exercise.

Contents

1 Executive Summary

All requirements for this assignment have been satisfied. A formulated proof is given for the derived inference rule for the accescc control scheme described in problem 1. In addition, responses are given for the encryption scheme for question 2 and the question on theroem provers and aritificial intelligence of question 3.

2 Question 1

2.1 Problem Statement

In this access control scheme, Alice and Bob take on the roles of Employee and Relay respectively. An inference rule is defined for the Relay to follow whenever it receives a message. If the conditions in the inference rule is met, Bob will encrypt a launch message with his key and send it out as the Relay. I assume a certificate authority is not needed in this access control scenario since Alice and Bob will be part of the same organization so excanging public keys legitimately should not be as much of an isssue.

The Inference rule is below.

```
Role Employee controls prop go ==>
Alice Reps Employee on go ==>
Key Alice speaks_for Alice ==>
Key Alice says prop go ==>
prop go impf prop launch ==>
Key Bob quoting Role Relay says prop launch
```

HOL datatypes and theorems for problem 1 below.

commands = go | launch

keyPrinc = Staff people | Role roles | Ap num

people = Alice | Bob

principals = PR keyPrinc | Key keyPrinc

roles = Employee | Relay

[RelayRuleLaunch]

```
⊢ (M,Oi,Os) sat Name (PR (Role Employee)) controls prop go ⇒
  (M,Oi,Os) sat
  reps (Name (PR (Staff Alice))) (Name (PR (Role Employee)))
    (prop go) ⇒
  (M,Oi,Os) sat
  Name (Key (Staff Alice)) quoting
  Name (PR (Role Employee)) says prop go ⇒
  (M,Oi,Os) sat prop go impf prop launch ⇒
  (M,Oi,Os) sat
  Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice)) ⇒
  (M,Oi,Os) sat
  Name (Key (Staff Bob)) quoting Name (PR (Role Relay)) says
  prop launch
```

2.2 Relevant Code

The proof of the derived inference rule involves using the *PAT_ASSUM* to inject hyptothesises into the proof until the goal is derived. The main theories used in the tactics are produced using the assumptions and the *CONTROLS*, *QUOTING*, *REPS*, and *SPEAKS_FOR* inference rules.

```
val RelayRuleLaunch =
TAC_PROOF(
([],
``(M,Oi,Os) sat Name (PR (Role Employee)) controls prop go ==>
  (M,Oi,Os) sat
  reps (Name (PR (Staff Alice))) (Name (PR (Role Employee)))
  (prop go) ==>
  (M,Oi,Os) sat
  Name (Key (Staff Alice)) quoting
  Name (PR (Role Employee)) says prop go ==>
  (M,Oi,Os) sat prop go impf prop launch ==>
  (M,Oi,Os) sat
  Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice)) ==>
  (M,Oi,Os) sat
  Name (Key (Staff Bob)) quoting Name (PR (Role Relay)) says
  prop launch``),
REPEAT STRIP_TAC THEN
ACL_SAYS_TAC THEN

PAT_ASSUM ``(M,Oi,Os) sat Name (Key (Staff Alice)) quoting
             Name (PR (Role Employee)) says prop go``
  (fn th => ASSUME_TAC (QUOTING_LR th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
             Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (Key (Staff Alice)) says
      Name (PR (Role Employee)) says (prop go)``
    (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat
             Name (PR (Staff Alice)) says Name (PR (Role Employee)) says
             (prop go)``
  (fn th => ASSUME_TAC (QUOTING_RL th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
            reps (Name (PR (Staff Alice))) (Name (PR (Role Employee)))(prop go)``
  (fn th1 =>
    (PAT_ASSUM ``(M,Oi,Os) sat
    Name (PR (Staff Alice)) quoting Name (PR (Role Employee)) says
    (prop go)``
      (fn th2 =>
        (PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role Employee)) controls (prop go)``
          (fn th3 => ASSUME_TAC (REPS th1 th2 th3)))))) THEN
```

```
PAT_ASSUM ``(M,Oi,Os) sat (prop go)`` (fn th1 =>
  (PAT_ASSUM ``(M,Oi,Os) sat prop go impf prop launch``
    (fn th2 => ASSUME_TAC (ACL_MP th1 th2)))) THEN

ASM_REWRITE_TAC[]);
```

2.3 Execution Transcript

```
---------------------------------------------------------------------
       HOL-4 [Kananaskis 11 (stdknl, built Sat Aug 19 09:30:06 2017)]

       For introductory HOL help, type: help "hol";
       To exit type <Control>-D
---------------------------------------------------------------------
[extending loadPath with Holmakefile INCLUDES variable]
> > > > Loading acl_infRules

> > > > > > > > > <<HOL message: Created theory "exam3">>
<<HOL message: Defined type: "commands">>
<<HOL message: Defined type: "roles">>
<<HOL message: Defined type: "people">>
<<HOL message: Defined type: "keyPrinc">>
<<HOL message: Defined type: "principals">>
<<HOL message: inventing new type variable names: 'a, 'b, 'c>>
<<HOL message: inventing new type variable names: 'a, 'b, 'c>>
<<HOL message: inventing new type variable names: 'a, 'b, 'c, 'd>>
<<HOL message: inventing new type variable names: 'a, 'b, 'c>>
<<HOL message: inventing new type variable names: 'a, 'b, 'c>>
<<HOL message: inventing new type variable names: 'a, 'b, 'c, 'd>>
Theory: exam3

Parents:
    aclDrules
    cipher

Type constants:
    commands 0
    keyPrinc 0
    people 0
    principals 0
    roles 0

Term constants:
    Alice              :people
    Ap                 :num -> keyPrinc
    Bob                :people
    Employee           :roles
```

```
    Key               :keyPrinc -> principals
    PR                :keyPrinc -> principals
    Relay             :roles
    Role              :roles -> keyPrinc
    Staff             :people -> keyPrinc
    commands2num      :commands -> num
    commands_CASE     :commands ->  ->  ->
    commands_size     :commands -> num
    go                :commands
    keyPrinc_CASE     :keyPrinc ->
                       (people -> ) -> (roles -> ) -> (num -> ) ->
    keyPrinc_size     :keyPrinc -> num
    launch            :commands
    num2commands      :num -> commands
    num2people        :num -> people
    num2roles         :num -> roles
    people2num        :people -> num
    people_CASE       :people ->  ->  ->
    people_size       :people -> num
    principals_CASE   :principals ->
                       (keyPrinc -> ) -> (keyPrinc -> ) ->
    principals_size   :principals -> num
    roles2num         :roles -> num
    roles_CASE        :roles ->  ->  ->
    roles_size        :roles -> num

Definitions:
    @tempAlice_def
      |- Alice = num2people 0
    @tempBob_def
      |- Bob = num2people 1
    @tempEmployee_def
      |- Employee = num2roles 0
    @tempRelay_def
      |- Relay = num2roles 1
    @tempgo_def
      |- go = num2commands 0
    @templaunch_def
      |- launch = num2commands 1
    commands_BIJ
      |- (a. num2commands (commands2num a) = a)
         r. (n. n < 2) r  (commands2num (num2commands r) = r)
    commands_CASE
      |- x v0 v1.
           (case x of go => v0 | launch => v1) =
           (m. if m = 0 then v0 else v1) (commands2num x)
    commands_TY_DEF
```

```
  |- rep. TYPE_DEFINITION (n. n < 2) rep
commands_size_def
  |- x. commands_size x = 0
keyPrinc_TY_DEF
  |- rep.
       TYPE_DEFINITION
         (a0.
            'keyPrinc' .
              (a0.
                 (a.
                    a0 =
                    (a.
                       ind_type$CONSTR 0 (a,ARB,ARB)
                         (n. ind_type$BOTTOM)) a)
                 (a.
                    a0 =
                    (a.
                       ind_type$CONSTR (SUC 0) (ARB,a,ARB)
                         (n. ind_type$BOTTOM)) a)
                 (a.
                    a0 =
                    (a.
                       ind_type$CONSTR (SUC (SUC 0)) (ARB,ARB,a)
                         (n. ind_type$BOTTOM)) a)
                 'keyPrinc' a0)
              'keyPrinc' a0) rep
keyPrinc_case_def
  |- (a f f1 f2. keyPrinc_CASE (Staff a) f f1 f2 = f a)
     (a f f1 f2. keyPrinc_CASE (Role a) f f1 f2 = f1 a)
     a f f1 f2. keyPrinc_CASE (Ap a) f f1 f2 = f2 a
keyPrinc_size_def
  |- (a. keyPrinc_size (Staff a) = 1 + people_size a)
     (a. keyPrinc_size (Role a) = 1 + roles_size a)
     a. keyPrinc_size (Ap a) = 1 + a
people_BIJ
  |- (a. num2people (people2num a) = a)
     r. (n. n < 2) r  (people2num (num2people r) = r)
people_CASE
  |- x v0 v1.
       (case x of Alice => v0 | Bob => v1) =
       (m. if m = 0 then v0 else v1) (people2num x)
people_TY_DEF
  |- rep. TYPE_DEFINITION (n. n < 2) rep
people_size_def
  |- x. people_size x = 0
principals_TY_DEF
  |- rep.
```

```
        TYPE_DEFINITION
          (a0.
             'principals' .
               (a0.
                  (a.
                     a0 =
                     (a. ind_type$CONSTR 0 a (n. ind_type$BOTTOM))
                       a)
                  (a.
                     a0 =
                     (a.
                        ind_type$CONSTR (SUC 0) a
                          (n. ind_type$BOTTOM)) a)
                  'principals' a0)
               'principals' a0) rep
    principals_case_def
      |- (a f f1. principals_CASE (PR a) f f1 = f a)
         a f f1. principals_CASE (Key a) f f1 = f1 a
    principals_size_def
      |- (a. principals_size (PR a) = 1 + keyPrinc_size a)
         a. principals_size (Key a) = 1 + keyPrinc_size a
    roles_BIJ
      |- (a. num2roles (roles2num a) = a)
         r. (n. n < 2) r  (roles2num (num2roles r) = r)
    roles_CASE
      |- x v0 v1.
           (case x of Employee => v0 | Relay => v1) =
           (m. if m = 0 then v0 else v1) (roles2num x)
    roles_TY_DEF
      |- rep. TYPE_DEFINITION (n. n < 2) rep
    roles_size_def
      |- x. roles_size x = 0

Theorems:
    RelayRuleLaunch
      |- (M,Oi,Os) sat Name (PR (Role Employee)) controls prop go
         (M,Oi,Os) sat
         reps (Name (PR (Staff Alice))) (Name (PR (Role Employee)))
           (prop go)
         (M,Oi,Os) sat
         Name (Key (Staff Alice)) quoting Name (PR (Role Employee)) says
         prop go
         (M,Oi,Os) sat prop go impf prop launch
         (M,Oi,Os) sat
         Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))
         (M,Oi,Os) sat
         Name (Key (Staff Bob)) quoting Name (PR (Role Relay)) says
```

```
     prop launch
commands2num_11
  |- a a'. (commands2num a = commands2num a')  (a = a')
commands2num_ONTO
  |- r. r < 2  a. r = commands2num a
commands2num_num2commands
  |- r. r < 2  (commands2num (num2commands r) = r)
commands2num_thm
  |- (commands2num go = 0)  (commands2num launch = 1)
commands_Axiom
  |- x0 x1. f. (f go = x0)  (f launch = x1)
commands_EQ_commands
  |- a a'. (a = a')  (commands2num a = commands2num a')
commands_case_cong
  |- M M' v0 v1.
       (M = M')  ((M' = go)  (v0 = v0'))
       ((M' = launch)  (v1 = v1'))
       ((case M of go => v0 | launch => v1) =
        case M' of go => v0' | launch => v1')
commands_case_def
  |- (v0 v1. (case go of go => v0 | launch => v1) = v0)
     v0 v1. (case launch of go => v0 | launch => v1) = v1
commands_distinct
  |- go  launch
commands_induction
  |- P. P go  P launch  a. P a
commands_nchotomy
  |- a. (a = go)  (a = launch)
datatype_commands
  |- DATATYPE (commands go launch)
datatype_keyPrinc
  |- DATATYPE (keyPrinc Staff Role Ap)
datatype_people
  |- DATATYPE (people Alice Bob)
datatype_principals
  |- DATATYPE (principals PR Key)
datatype_roles
  |- DATATYPE (roles Employee Relay)
keyPrinc_11
  |- (a a'. (Staff a = Staff a')  (a = a'))
     (a a'. (Role a = Role a')  (a = a'))
     a a'. (Ap a = Ap a')  (a = a')
keyPrinc_Axiom
  |- f0 f1 f2.
       fn.
         (a. fn (Staff a) = f0 a)  (a. fn (Role a) = f1 a)
         a. fn (Ap a) = f2 a
```

```
keyPrinc_case_cong
  |- M M' f f1 f2.
       (M = M')  (a. (M' = Staff a)  (f a = f' a))
       (a. (M' = Role a)  (f1 a = f1' a))
       (a. (M' = Ap a)  (f2 a = f2' a))
       (keyPrinc_CASE M f f1 f2 = keyPrinc_CASE M' f' f1' f2')
keyPrinc_distinct
  |- (a' a. Staff a  Role a')  (a' a. Staff a  Ap a')
     a' a. Role a  Ap a'
keyPrinc_induction
  |- P.
       (p. P (Staff p))  (r. P (Role r))  (n. P (Ap n))
       k. P k
keyPrinc_nchotomy
  |- kk. (p. kk = Staff p)  (r. kk = Role r)  n. kk = Ap n
num2commands_11
  |- r r'.
       r < 2
       r' < 2
       ((num2commands r = num2commands r')  (r = r'))
num2commands_ONTO
  |- a. r. (a = num2commands r)  r < 2
num2commands_commands2num
  |- a. num2commands (commands2num a) = a
num2commands_thm
  |- (num2commands 0 = go)  (num2commands 1 = launch)
num2people_11
  |- r r'.
       r < 2  r' < 2  ((num2people r = num2people r')  (r = r'))
num2people_ONTO
  |- a. r. (a = num2people r)  r < 2
num2people_people2num
  |- a. num2people (people2num a) = a
num2people_thm
  |- (num2people 0 = Alice)  (num2people 1 = Bob)
num2roles_11
  |- r r'.
       r < 2  r' < 2  ((num2roles r = num2roles r')  (r = r'))
num2roles_ONTO
  |- a. r. (a = num2roles r)  r < 2
num2roles_roles2num
  |- a. num2roles (roles2num a) = a
num2roles_thm
  |- (num2roles 0 = Employee)  (num2roles 1 = Relay)
people2num_11
  |- a a'. (people2num a = people2num a')  (a = a')
people2num_ONTO
```

```
  |- r. r < 2  a. r = people2num a
people2num_num2people
  |- r. r < 2  (people2num (num2people r) = r)
people2num_thm
  |- (people2num Alice = 0)  (people2num Bob = 1)
people_Axiom
  |- x0 x1. f. (f Alice = x0)  (f Bob = x1)
people_EQ_people
  |- a a'. (a = a')  (people2num a = people2num a')
people_case_cong
  |- M M' v0 v1.
       (M = M')  ((M' = Alice)  (v0 = v0'))
       ((M' = Bob)  (v1 = v1'))
       ((case M of Alice => v0 | Bob => v1) =
        case M' of Alice => v0' | Bob => v1')
people_case_def
  |- (v0 v1. (case Alice of Alice => v0 | Bob => v1) = v0)
     v0 v1. (case Bob of Alice => v0 | Bob => v1) = v1
people_distinct
  |- Alice  Bob
people_induction
  |- P. P Alice  P Bob  a. P a
people_nchotomy
  |- a. (a = Alice)  (a = Bob)
principals_11
  |- (a a'. (PR a = PR a')  (a = a'))
     a a'. (Key a = Key a')  (a = a')
principals_Axiom
  |- f0 f1. fn. (a. fn (PR a) = f0 a)  a. fn (Key a) = f1 a
principals_case_cong
  |- M M' f f1.
       (M = M')  (a. (M' = PR a)  (f a = f' a))
       (a. (M' = Key a)  (f1 a = f1' a))
       (principals_CASE M f f1 = principals_CASE M' f' f1')
principals_distinct
  |- a' a. PR a  Key a'
principals_induction
  |- P. (k. P (PR k))  (k. P (Key k))  p. P p
principals_nchotomy
  |- pp. (k. pp = PR k)  k. pp = Key k
roles2num_11
  |- a a'. (roles2num a = roles2num a')  (a = a')
roles2num_ONTO
  |- r. r < 2  a. r = roles2num a
roles2num_num2roles
  |- r. r < 2  (roles2num (num2roles r) = r)
roles2num_thm
```

```
      |- (roles2num Employee = 0)  (roles2num Relay = 1)
    roles_Axiom
      |- x0 x1. f. (f Employee = x0)  (f Relay = x1)
    roles_EQ_roles
      |- a a'. (a = a')  (roles2num a = roles2num a')
    roles_case_cong
      |- M M' v0 v1.
           (M = M')  ((M' = Employee)  (v0 = v0'))
           ((M' = Relay)  (v1 = v1'))
           ((case M of Employee => v0 | Relay => v1) =
            case M' of Employee => v0' | Relay => v1')
    roles_case_def
      |- (v0 v1.
            (case Employee of Employee => v0 | Relay => v1) = v0)
         v0 v1. (case Relay of Employee => v0 | Relay => v1) = v1
    roles_distinct
      |- Employee  Relay
    roles_induction
      |- P. P Employee  P Relay  a. P a
    roles_nchotomy
      |- a. (a = Employee)  (a = Relay)
Exporting theory "exam3" ... done.
Theory "exam3" took 0.70370s to build
structure exam3Script:
  sig

  end
val it = (): unit
>
*** Emacs/HOL command completed ***

>
```

3 Question 2

Brian want to communicate securely using a different symmetric key for every message sent. In addition, they must follow corporate policy and have all messages be transferred using a secure relay. To accomplish this, they first get a set of n keys from the TA authority which also gives the keys to the secure relay.

The set of keys for Alice is KA1,KA2,....KAn with published certificate encrypt(Kta, [KA1,KA2...KAN]Alice) Not explicitely stated above but key identifiers for ALice and the relay will be also be sent and included in the certificate. Similar keys and certificate will be produced and given to Bob. In this scheme, Bob and Alice will exchange their key identifiers with each other.

To initiate the secure channel, Alice will send her key hint pair(for key Ka), Bob's published certificate, as well as a key indentifier for Bob(Kb) to the relay. Subsequent messages to the Relay will feature a different set of key identifiers that were received from the trusted authority. Multiple keys from the TA were needed since if only one was used by both Alice and Bob to communicate with the relay, then if that key was exposed an attacker could uncover every shared key generated by the relay for Alice and Bob to use.

The relay will use the TA's key in order to verify that the certificate and the sent identifier belongs to Bob. Once it verifies the ceritificate and key identifier for Kb is Bob's, it will generate a new key and send the identifier *KB = encrypt(Kb,K)* as well as identifier for Kb to Bob. It will also send *KA = encrypt(Ka,K)* to Alice with the identifier for Ka. Alice and Bob can now get the shared key K by decrypting with their respective keys and Alice can send an encrypted message to Bob that he can decrypt with K.

To authenticate that the message came from Alice, Alice will neeed to have sent her TA certiificate with the relay with the rest of the initial message sent to the relay. The relay will decrypt using Kta then it will know that the key identifier Ka belongs to Alice. It can then generate the certificate *encrypt(Kb,[KB,KA,Alice])* paired with Kb's indentifier. When Bob gets this certificate, he can verify with his key that the key identifier KB (as well as the new key K) was initiated from Alice since he trusts the relay and the TA. He therefore knows messages that he can decrypt with K originated from Alice.

4 Question 3

To facilitate the use of theorem provers like HOL in artiificial intelligence, the intelligence will need to be build on verly large libraries made initially by people. Due to the difficulty in desigining automated proofs for higher order logic sytems, very extensive libraries possibly made with the assistance of ATPs will be needed. In addition, there would need to be a common ground and methodology established between all of the theorem provers. Without a general model provided, it will be hard for artiificial intelligence algorithms to work will all the different theorem provers.

A Source For exam3Script.sml

The following code is from *HOL/exam3Script.sml*

```
structure exam3Script = struct

(* only necessary when working interactively
app load ["acl_infRules","aclrulesTheory","aclDrulesTheory","exam3Theory", "cipherThe
open acl_infRules aclrulesTheory aclDrulesTheory cipherTheory exam3Theory
*)

open HolKernel boolLib Parse bossLib
open acl_infRules aclrulesTheory aclDrulesTheory cipherTheory

val _ = new_theory "exam3"

val _ =
Datatype
`commands = go | launch`

val _ =
Datatype
`roles = Employee | Relay`

val _ =
Datatype
`people = Alice | Bob`

val _ =
Datatype
`keyPrinc = Staff exam3$people | Role exam3$roles | Ap num`

val _ =
Datatype
`principals = PR keyPrinc | Key keyPrinc`

(* Relay derived inference rule

   Role Employee controls prop go ==>
   Alice Reps Employee on go ==>
   Key Alice speaks_for Alice ==>
   Key Alice says prop go ==>
   prop go impf prop launch ==>
   Key Bob quoting Role Relay says prop launch


set_goal
([],
```

```
``(M,Oi,Os) sat Name (PR (Role Employee)) controls prop go ==>
  (M,Oi,Os) sat
  reps (Name (PR (Staff Alice))) (Name (PR (Role Employee)))
  (prop go) ==>
  (M,Oi,Os) sat
  Name (Key (Staff Alice)) quoting
  Name (PR (Role Employee)) says prop go ==>
  (M,Oi,Os) sat prop go impf prop launch ==>
  (M,Oi,Os) sat
  Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice)) ==>
  (M,Oi,Os) sat
  Name (Key (Staff Bob)) quoting Name (PR (Role Relay)) says
  prop launch``)
REPEAT STRIP_TAC THEN
ACL_SAYS_TAC THEN

PAT_ASSUM ``(M,Oi,Os) sat Name (Key (Staff Alice)) quoting
           Name (PR (Role Employee)) says prop go``
  (fn th => ASSUME_TAC (QUOTING_LR th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
           Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (Key (Staff Alice)) says
      Name (PR (Role Employee)) says (prop go)``
    (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat
           Name (PR (Staff Alice)) says Name (PR (Role Employee)) says
           (prop go)``
  (fn th => ASSUME_TAC (QUOTING_RL th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
          reps (Name (PR (Staff Alice))) (Name (PR (Role Employee)))(prop go)``
  (fn th1 =>
    (PAT_ASSUM ``(M,Oi,Os) sat
    Name (PR (Staff Alice)) quoting Name (PR (Role Employee)) says
    (prop go)``
      (fn th2 =>
        (PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role Employee)) controls (prop go)`
          (fn th3 => ASSUME_TAC (REPS th1 th2 th3)))))) THEN

PAT_ASSUM ``(M,Oi,Os) sat (prop go)`` (fn th1 =>
  (PAT_ASSUM ``(M,Oi,Os) sat prop go impf prop launch``
    (fn th2 => ASSUME_TAC (ACL_MP th1 th2)))) THEN
```

```
ASM_REWRITE_TAC[]

*)

val RelayRuleLaunch =
TAC_PROOF(
([],
``(M,Oi,Os) sat Name (PR (Role Employee)) controls prop go ==>
  (M,Oi,Os) sat
  reps (Name (PR (Staff Alice))) (Name (PR (Role Employee)))
  (prop go) ==>
  (M,Oi,Os) sat
  Name (Key (Staff Alice)) quoting
  Name (PR (Role Employee)) says prop go ==>
  (M,Oi,Os) sat prop go impf prop launch ==>
  (M,Oi,Os) sat
  Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice)) ==>
  (M,Oi,Os) sat
  Name (Key (Staff Bob)) quoting Name (PR (Role Relay)) says
  prop launch``),
REPEAT STRIP_TAC THEN
ACL_SAYS_TAC THEN

PAT_ASSUM ``(M,Oi,Os) sat Name (Key (Staff Alice)) quoting
           Name (PR (Role Employee)) says prop go``
  (fn th => ASSUME_TAC (QUOTING_LR th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
           Name (Key (Staff Alice)) speaks_for Name (PR (Staff Alice))``
  (fn th1 =>
    (PAT_ASSUM
    ``(M,Oi,Os) sat Name (Key (Staff Alice)) says
      Name (PR (Role Employee)) says (prop go)``
    (fn th2 => ASSUME_TAC(SPEAKS_FOR th1 th2)))) THEN

PAT_ASSUM ``(M,Oi,Os) sat
           Name (PR (Staff Alice)) says Name (PR (Role Employee)) says
           (prop go)``
  (fn th => ASSUME_TAC (QUOTING_RL th)) THEN

PAT_ASSUM ``(M,Oi,Os) sat
          reps (Name (PR (Staff Alice))) (Name (PR (Role Employee)))(prop go)``
  (fn th1 =>
    (PAT_ASSUM ``(M,Oi,Os) sat
    Name (PR (Staff Alice)) quoting Name (PR (Role Employee)) says
    (prop go)``
      (fn th2 =>
```

```
      (PAT_ASSUM ``(M,Oi,Os) sat Name (PR (Role Employee)) controls (prop go)`
        (fn th3 => ASSUME_TAC (REPS th1 th2 th3)))))) THEN

PAT_ASSUM ``(M,Oi,Os) sat (prop go)`` (fn th1 =>
  (PAT_ASSUM ``(M,Oi,Os) sat prop go impf prop launch``
    (fn th2 => ASSUME_TAC (ACL_MP th1 th2)))) THEN

ASM_REWRITE_TAC[]);

val _ = save_thm("RelayRuleLaunch",RelayRuleLaunch)

val _ = print_theory "-";

val _ = export_theory();

end (* structure *)
```