Exam 4 Report

Alfred Murabito

March 27, 2020

Abstract

This report answers the three questions in the CIS 634 Exam 4. There are theories proved for questions 1 and 2 and an opinioned response for policy spaces in question 3.

Acknowledgments: I received no assistance with this exercise.

Contents

1 Executive Summary

All requirements for this assignment have been satisfied. The first question is answered in the code in the HOL/exam4Script.sml source file. There are four derived inference rules for all the access control scheme of the large distributed network The simple opener state machine theory described in problem 2 can be found in simpleOpener.Script. Question 3 involves my opinion on policy spaces. 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

The following problem involves an access control scheme controlling authorization and access to a non-local filesystem on a very large distributed network. The proof below is for a user Ursala requesting to have a file printed. Access control to the filesystem on the network is controlled by three servers AS(Authentication Server), TGS(Ticket Granting Server), and the FS (File Server).

1. Request a ticket bearing ticket from the AS, which the AS will respond with the Ticket Bearing Ticket and encrypted session key
2. The user's client program will receive the session key and ticket, and make a request to get a ticket for the FS server using these items.
3. Ursala will send a message, encrypted with the TGS session key, to the TGS server as well as the Ticket Bearing Ticket, and then receive a service granting ticket to be sent to the FS server with the new session key to be used with FS as well..
4. Ursala will send a new message to the FS encrypted with the latest session key that requests access to print the file along with the service granting ticket from before. If Ursala is authorized to print the file, the FS will honor the request.

HOL datatypes for probelm one below.

access = print | read | write

keys = KTGS | KFS | KU | KUTGS | KUFS

people = Ursala

princs = PR servers | User people | Key keys

props = USE serviceKey | PK people keys | AC access

servers = AS | TGS | FS

serviceKey = SERV servers | K keys

2.2 Relevant Code

Part A. Inference rule for the AS authenticating Ursala's request to use the TGS service. The ticket granting ticket is represented here by the Name (Key KTGS) says prop (PK Ursala KUTGS)) formula. The ticket is the (PK Ursala KUTGS) proposition, and the (Key KTGS) says prefix means that it is encrypted with the TGS key, which is only available to the AS and TGS servers.

```
⊢ (M,Oi,Os) sat Name (User Ursala) says prop (USE (SERV TGS)) ⇒
  (M,Oi,Os) sat
  prop (USE (SERV TGS)) impf
  Name (Key KU) says prop (USE (K KUTGS)) andf
  Name (Key KTGS) says prop (PK Ursala KUTGS) ⇒
  (M,Oi,Os) sat
  Name (User Ursala) controls prop (USE (SERV TGS)) ⇒
  (M,Oi,Os) sat
  Name (Key KU) says prop (USE (K KUTGS)) andf
  Name (Key KTGS) says prop (PK Ursala KUTGS)
```

```
val forma1 = ``Name (User Ursala) says prop (USE (SERV TGS)):(props,princs,'d,'e)Form
val forma2 = ``Name (User Ursala) controls prop (USE (SERV TGS)):(props,princs,'d,'e)
val forma3 = ``prop (USE (SERV TGS)) impf (Name (Key KU) says prop (USE (K KUTGS))
                                                                        andf Name
val [forma1_thm,forma2_thm,forma3_thm] = map ACL_ASSUM [forma1,forma2,forma3]

val forma4_thm = CONTROLS forma2_thm forma1_thm
val init_auth_thm = ACL_MP forma4_thm forma3_thm
```

Part B: The inference for part B deals with the network client on Ursala's machine making the decision to use the session key that it obtains from the message from AS that was encrypted with Ursala's and the AS's shared key. After deciding to use the session key (Key KUTGS), Ursala will send the encrypted request to use the FS server, symbolized by Name (Key KUTGS) says prop (USE (SERV FS)), and also the Ticket Bearing Ticket encrytped with (Key KTGS) from before.

```
⊢ (M,Oi,Os) sat Name (Key KU) speaks_for Name (PR AS) ⇒
  (M,Oi,Os) sat
  prop (USE (K KUTGS)) impf
  Name (Key KUTGS) says prop (USE (SERV FS)) ⇒
  (M,Oi,Os) sat Name (PR AS) controls prop (USE (K KUTGS)) ⇒
  (M,Oi,Os) sat
  Name (Key KU) says prop (USE (K KUTGS)) andf
  Name (Key KTGS) says prop (PK Ursala KUTGS) ⇒
  (M,Oi,Os) sat
  Name (Key KUTGS) says prop (USE (SERV FS)) andf
  Name (Key KTGS) says prop (PK Ursala KUTGS)
```

```
val formb1 = ``(Name (Key KU) says prop (USE (K KUTGS)) andf
               Name (Key KTGS) says prop (PK Ursala KUTGS)):(props,princs,'d,'e)F

val formb2 = ``Name (PR AS) controls prop (USE (K KUTGS)):(props,princs,'d,'e)For
val formb3 = ``(Name (Key KU) speaks_for Name (PR AS)):(props,princs,'d,'e)Form`
val formb4 = ``prop (USE (K KUTGS)) impf Name (Key KUTGS) says prop (USE (SERV FS

val [formb1_thm,formb2_thm,formb3_thm,formb4_thm] = map ACL_ASSUM[formb1,formb2,f
val formb5_thm = ACL_SIMP1 formb1_thm
val formb6_thm = ACL_SIMP2 formb1_thm
val formb7_thm = SPEAKS_FOR formb3_thm formb5_thm
val formb8_thm = CONTROLS formb2_thm formb7_thm
val formb9_thm = ACL_MP formb8_thm formb4_thm
val session_key_receipt_thm = ACL_CONJ formb9_thm formb6_thm
```

Part C: The next inference rule proved is the decision to grant the request for services ticket, symbolized by formula Name (Key KFS) says ((Name (Key KUFS)) speaks_for (Name (User Ursala))). This message is encyrpted with the key only known by the TGS and FS servers, and it tells the FS server that the session key KUFS belongs to the user Ursala. The TGS wil also generate a session key KUFS, which is encrypted with KUTGS then sent.

The decision to grant these items is dependent on the TGS able to decrypt the ticket (PK Ursala KUTGS) which says that the key KUTGS is valid. Once that key is decided to be valid, the request USE SERV FS which was also sent can be deemed valid since it was encrypted with KUTGS, which should only be known to Ursala and AS.

Note, the formula (Name (Key KUFS) says prop (AC read) is derived from a valid ticket because the User Ursala will have access to the session key KUFS generated and can make this request to the final server.

```
⊢ (M,Oi,Os) sat Name (Key KTGS) speaks_for Name (PR AS) ⇒
  (M,Oi,Os) sat
  prop (PK Ursala KUTGS) impf
  Name (Key KUFS) says prop (AC read) ⇒
  (M,Oi,Os) sat
  prop (PK Ursala KUTGS) impf
  Name (Key KFS) says
  Name (Key KUFS) speaks_for Name (User Ursala) ⇒
  (M,Oi,Os) sat Name (PR AS) controls prop (PK Ursala KUTGS) ⇒
  (M,Oi,Os) sat
  Name (Key KUTGS) says prop (USE (SERV FS)) andf
  Name (Key KTGS) says prop (PK Ursala KUTGS) ⇒
  (M,Oi,Os) sat
  Name (Key KFS) says
  Name (Key KUFS) speaks_for Name (User Ursala) andf
  Name (Key KUFS) says prop (AC read)
```

```
val formc1 = ``Name (Key KUTGS) says prop (USE (SERV FS)) andf
               Name (Key KTGS) says prop (PK Ursala KUTGS):(props,princs,'d,'e)Form``

val formc2 = ``(Name (PR AS)) controls (prop (PK Ursala KUTGS)):(props,princs,'d,'e)F
val formc3 = ``(Name (Key KTGS) speaks_for (Name (PR AS))):(props,princs,'d,'e)Form``

val formc4 = ``prop (PK Ursala KUTGS) impf (Name (Key KFS) says ((Name (Key KUFS)) sp
val formc5 = ``prop (PK Ursala KUTGS) impf (Name (Key KUFS) says prop (AC read)):(pro

val [formc1_thm,formc2_thm,formc3_thm,formc4_thm,formc5_thm] = map ACL_ASSUM[formc1,f

val formc6_thm = ACL_SIMP1 formc1_thm
val formc7_thm = ACL_SIMP2 formc1_thm
val formc8_thm = SPEAKS_FOR formc3_thm formc7_thm
val formc9_thm = CONTROLS formc2_thm formc8_thm
val formc10_thm = ACL_MP formc9_thm formc4_thm
val formc11_thm = ACL_MP formc9_thm formc5_thm
val request_for_services_thm = ACL_CONJ formc10_thm formc11_thm
```

Part D: The final inference rule proof involes the FS server granting Ursala's request to print. The FS authenticated the request using the request ticket sent to it by Ursala. It encovers the ticket by decrypting with the shared key KFS, then knows that the session key KUFS is valid to represent Ursala. Since the message requesting prniting was encyrpted with this key KUFS, it knows Alice must have made the request.

```
⊢ (M,Oi,Os) sat Name (Key KFS) speaks_for Name (PR TGS) ⇒
  (M,Oi,Os) sat Name (User Ursala) controls prop (AC print) ⇒
  (M,Oi,Os) sat
  Name (PR TGS) controls
  Name (Key KUFS) speaks_for Name (User Ursala) ⇒
  (M,Oi,Os) sat
  Name (Key KFS) says
  Name (Key KUFS) speaks_for Name (User Ursala) andf
  Name (Key KUFS) says prop (AC print) ⇒
  (M,Oi,Os) sat prop (AC print)
```

```
val formd1 = ``(Name (Key KFS) says (Name (Key KUFS) speaks_for Name (User Ursala
                                (Name (Key KUFS) says prop (AC print)):(props,princs,
val formd2 = ``(Name (Key KFS) speaks_for Name (PR TGS)):(props,princs,'d,'e)Form
val formd3 = ``(Name (PR TGS) controls (Name (Key KUFS) speaks_for Name (User Ur
val formd4 = ``(Name (User Ursala) controls prop(AC print)):(props,princs,'d,'e)

val [formd1_thm,formd2_thm,formd3_thm,formd4_thm] = map ACL_ASSUM[formd1,formd2,
val formd5_thm = ACL_SIMP1 formd1_thm
val formd6_thm = ACL_SIMP2 formd1_thm
val formd7_thm = SPEAKS_FOR formd2_thm formd5_thm
val formd8_thm = CONTROLS formd3_thm formd7_thm
val formd9_thm = SPEAKS_FOR formd8_thm formd6_thm
val service_request_thm = CONTROLS formd4_thm formd9_thm
```

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 "exam4">>
<<HOL message: Defined type: "servers">>
<<HOL message: Defined type: "people">>
<<HOL message: Defined type: "keys">>
<<HOL message: Defined type: "princs">>
<<HOL message: Defined type: "access">>
```

```
<<HOL message: Defined type: "serviceKey">>
<<HOL message: Defined type: "props">>
Theory: exam4

Parents:
    aclDrules

Type constants:
    access 0
    keys 0
    people 0
    princs 0
    props 0
    servers 0
    serviceKey 0

Term constants:
    AC               :access -> props
    AS               :servers
    FS               :servers
    K                :keys -> serviceKey
    KFS              :keys
    KTGS             :keys
    KU               :keys
    KUFS             :keys
    KUTGS            :keys
    Key              :keys -> princs
    PK               :people -> keys -> props
    PR               :servers -> princs
    SERV             :servers -> serviceKey
    TGS              :servers
    USE              :serviceKey -> props
    Ursala           :people
    User             :people -> princs
    access2num       :access -> num
    access_CASE      :access ->  ->  ->  ->
    access_size      :access -> num
    keys2num         :keys -> num
    keys_CASE        :keys ->  ->  ->  ->  ->  ->
    keys_size        :keys -> num
    num2access       :num -> access
    num2keys         :num -> keys
    num2people       :num -> people
    num2servers      :num -> servers
    people2num       :people -> num
    people_CASE      :people ->  ->
    people_size      :people -> num
```

```
    princs_CASE       :princs ->
                       (servers -> ) ->
                       (people -> ) -> (keys -> ) ->
    princs_size       :princs -> num
    print             :access
    props_CASE        :props ->
                       (serviceKey -> ) ->
                       (people -> keys -> ) -> (access -> ) ->
    props_size        :props -> num
    read              :access
    servers2num       :servers -> num
    servers_CASE      :servers ->  ->  ->  ->
    servers_size      :servers -> num
    serviceKey_CASE   :serviceKey -> (servers -> ) -> (keys -> ) ->
    serviceKey_size   :serviceKey -> num
    write             :access

Definitions:
    @tempAS_def
      |- AS = num2servers 0
    @tempFS_def
      |- FS = num2servers 2
    @tempKFS_def
      |- KFS = num2keys 1
    @tempKTGS_def
      |- KTGS = num2keys 0
    @tempKUFS_def
      |- KUFS = num2keys 4
    @tempKUTGS_def
      |- KUTGS = num2keys 3
    @tempKU_def
      |- KU = num2keys 2
    @tempTGS_def
      |- TGS = num2servers 1
    @tempUrsala_def
      |- Ursala = num2people 0
    @tempprint_def
      |- print = num2access 0
    @tempread_def
      |- read = num2access 1
    @tempwrite_def
      |- write = num2access 2
    access_BIJ
      |- (a. num2access (access2num a) = a)
         r. (n. n < 3) r  (access2num (num2access r) = r)
    access_CASE
      |- x v0 v1 v2.
```

```
       (case x of print => v0 | read => v1 | write => v2) =
       (m. if m < 1 then v0 else if m = 1 then v1 else v2)
         (access2num x)
access_TY_DEF
  |- rep. TYPE_DEFINITION (n. n < 3) rep
access_size_def
  |- x. access_size x = 0
keys_BIJ
  |- (a. num2keys (keys2num a) = a)
     r. (n. n < 5) r  (keys2num (num2keys r) = r)
keys_CASE
  |- x v0 v1 v2 v3 v4.
       (case x of
          KTGS => v0
        | KFS => v1
        | KU => v2
        | KUTGS => v3
        | KUFS => v4) =
       (m.
          if m < 2 then if m = 0 then v0 else v1
          else if m < 3 then v2
          else if m = 3 then v3
          else v4) (keys2num x)
keys_TY_DEF
  |- rep. TYPE_DEFINITION (n. n < 5) rep
keys_size_def
  |- x. keys_size x = 0
people_BIJ
  |- (a. num2people (people2num a) = a)
     r. (n. n < 1) r  (people2num (num2people r) = r)
people_CASE
  |- x v0. (case x of Ursala => v0) = (m. v0) (people2num x)
people_TY_DEF
  |- rep. TYPE_DEFINITION (n. n < 1) rep
people_size_def
  |- x. people_size x = 0
princs_TY_DEF
  |- rep.
       TYPE_DEFINITION
         (a0.
            'princs' .
              (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)
                  'princs' a0)
               'princs' a0) rep
princs_case_def
  |- (a f f1 f2. princs_CASE (PR a) f f1 f2 = f a)
     (a f f1 f2. princs_CASE (User a) f f1 f2 = f1 a)
     a f f1 f2. princs_CASE (Key a) f f1 f2 = f2 a
princs_size_def
  |- (a. princs_size (PR a) = 1 + servers_size a)
     (a. princs_size (User a) = 1 + people_size a)
     a. princs_size (Key a) = 1 + keys_size a
props_TY_DEF
  |- rep.
       TYPE_DEFINITION
         (a0'.
            'props' .
              (a0'.
                 (a.
                    a0' =
                    (a.
                       ind_type$CONSTR 0 (a,ARB,ARB,ARB)
                         (n. ind_type$BOTTOM)) a)
                 (a0 a1.
                    a0' =
                    (a0 a1.
                       ind_type$CONSTR (SUC 0) (ARB,a0,a1,ARB)
                         (n. ind_type$BOTTOM)) a0 a1)
                 (a.
                    a0' =
                    (a.
                       ind_type$CONSTR (SUC (SUC 0)) (ARB,ARB,ARB,a)
                         (n. ind_type$BOTTOM)) a)
                 'props' a0')
               'props' a0') rep
props_case_def
  |- (a f f1 f2. props_CASE (USE a) f f1 f2 = f a)
     (a0 a1 f f1 f2. props_CASE (PK a0 a1) f f1 f2 = f1 a0 a1)
     a f f1 f2. props_CASE (AC a) f f1 f2 = f2 a
```

```
    props_size_def
      |- (a. props_size (USE a) = 1 + serviceKey_size a)
         (a0 a1.
            props_size (PK a0 a1) =
            1 + (people_size a0 + keys_size a1))
         a. props_size (AC a) = 1 + access_size a
    servers_BIJ
      |- (a. num2servers (servers2num a) = a)
         r. (n. n < 3) r  (servers2num (num2servers r) = r)
    servers_CASE
      |- x v0 v1 v2.
           (case x of AS => v0 | TGS => v1 | FS => v2) =
           (m. if m < 1 then v0 else if m = 1 then v1 else v2)
             (servers2num x)
    servers_TY_DEF
      |- rep. TYPE_DEFINITION (n. n < 3) rep
    servers_size_def
      |- x. servers_size x = 0
    serviceKey_TY_DEF
      |- rep.
           TYPE_DEFINITION
             (a0.
                'serviceKey' .
                  (a0.
                     (a.
                        a0 =
                        (a.
                           ind_type$CONSTR 0 (a,ARB)
                             (n. ind_type$BOTTOM)) a)
                     (a.
                        a0 =
                        (a.
                           ind_type$CONSTR (SUC 0) (ARB,a)
                             (n. ind_type$BOTTOM)) a)
                     'serviceKey' a0)
                'serviceKey' a0) rep
    serviceKey_case_def
      |- (a f f1. serviceKey_CASE (SERV a) f f1 = f a)
         a f f1. serviceKey_CASE (K a) f f1 = f1 a
    serviceKey_size_def
      |- (a. serviceKey_size (SERV a) = 1 + servers_size a)
         a. serviceKey_size (K a) = 1 + keys_size a

Theorems:
    access2num_11
      |- a a'. (access2num a = access2num a')  (a = a')
    access2num_ONTO
```

```
  |- r. r < 3  a. r = access2num a
access2num_num2access
  |- r. r < 3  (access2num (num2access r) = r)
access2num_thm
  |- (access2num print = 0)  (access2num read = 1)
     (access2num write = 2)
access_Axiom
  |- x0 x1 x2. f. (f print = x0)  (f read = x1)  (f write = x2)
access_EQ_access
  |- a a'. (a = a')  (access2num a = access2num a')
access_case_cong
  |- M M' v0 v1 v2.
       (M = M')  ((M' = print)  (v0 = v0'))
       ((M' = read)  (v1 = v1'))  ((M' = write)  (v2 = v2'))
       ((case M of print => v0 | read => v1 | write => v2) =
        case M' of print => v0' | read => v1' | write => v2')
access_case_def
  |- (v0 v1 v2.
        (case print of print => v0 | read => v1 | write => v2) =
        v0)
     (v0 v1 v2.
        (case read of print => v0 | read => v1 | write => v2) =
        v1)
     v0 v1 v2.
       (case write of print => v0 | read => v1 | write => v2) = v2
access_distinct
  |- print  read  print  write  read  write
access_induction
  |- P. P print  P read  P write  a. P a
access_nchotomy
  |- a. (a = print)  (a = read)  (a = write)
datatype_access
  |- DATATYPE (access print read write)
datatype_keys
  |- DATATYPE (keys KTGS KFS KU KUTGS KUFS)
datatype_people
  |- DATATYPE (people Ursala)
datatype_princs
  |- DATATYPE (princs PR User Key)
datatype_props
  |- DATATYPE (props USE PK AC)
datatype_servers
  |- DATATYPE (servers AS TGS FS)
datatype_serviceKey
  |- DATATYPE (serviceKey SERV K)
init_auth_thm
  |- (M,Oi,Os) sat Name (User Ursala) says prop (USE (SERV TGS))
```

```
     (M,Oi,Os) sat
     prop (USE (SERV TGS)) impf
     Name (Key KU) says prop (USE (K KUTGS)) andf
     Name (Key KTGS) says prop (PK Ursala KUTGS)
     (M,Oi,Os) sat
     Name (User Ursala) controls prop (USE (SERV TGS))
     (M,Oi,Os) sat
     Name (Key KU) says prop (USE (K KUTGS)) andf
     Name (Key KTGS) says prop (PK Ursala KUTGS)
keys2num_11
  |- a a'. (keys2num a = keys2num a')  (a = a')
keys2num_ONTO
  |- r. r < 5  a. r = keys2num a
keys2num_num2keys
  |- r. r < 5  (keys2num (num2keys r) = r)
keys2num_thm
  |- (keys2num KTGS = 0)  (keys2num KFS = 1)  (keys2num KU = 2)
     (keys2num KUTGS = 3)  (keys2num KUFS = 4)
keys_Axiom
  |- x0 x1 x2 x3 x4.
       f.
         (f KTGS = x0)  (f KFS = x1)  (f KU = x2)
         (f KUTGS = x3)  (f KUFS = x4)
keys_EQ_keys
  |- a a'. (a = a')  (keys2num a = keys2num a')
keys_case_cong
  |- M M' v0 v1 v2 v3 v4.
       (M = M')  ((M' = KTGS)  (v0 = v0'))
       ((M' = KFS)  (v1 = v1'))  ((M' = KU)  (v2 = v2'))
       ((M' = KUTGS)  (v3 = v3'))  ((M' = KUFS)  (v4 = v4'))
       ((case M of
           KTGS => v0
         | KFS => v1
         | KU => v2
         | KUTGS => v3
         | KUFS => v4) =
        case M' of
          KTGS => v0'
        | KFS => v1'
        | KU => v2'
        | KUTGS => v3'
        | KUFS => v4')
keys_case_def
  |- (v0 v1 v2 v3 v4.
        (case KTGS of
           KTGS => v0
         | KFS => v1
```

```
          | KU => v2
          | KUTGS => v3
          | KUFS => v4) =
         v0)
      (v0 v1 v2 v3 v4.
         (case KFS of
            KTGS => v0
          | KFS => v1
          | KU => v2
          | KUTGS => v3
          | KUFS => v4) =
         v1)
      (v0 v1 v2 v3 v4.
         (case KU of
            KTGS => v0
          | KFS => v1
          | KU => v2
          | KUTGS => v3
          | KUFS => v4) =
         v2)
      (v0 v1 v2 v3 v4.
         (case KUTGS of
            KTGS => v0
          | KFS => v1
          | KU => v2
          | KUTGS => v3
          | KUFS => v4) =
         v3)
      v0 v1 v2 v3 v4.
        (case KUFS of
           KTGS => v0
         | KFS => v1
         | KU => v2
         | KUTGS => v3
         | KUFS => v4) =
        v4
keys_distinct
  |- KTGS  KFS  KTGS  KU  KTGS  KUTGS  KTGS  KUFS
     KFS  KU  KFS  KUTGS  KFS  KUFS  KU  KUTGS  KU  KUFS
     KUTGS  KUFS
keys_induction
  |- P. P KFS  P KTGS  P KU  P KUFS  P KUTGS  a. P a
keys_nchotomy
  |- a.
       (a = KTGS)  (a = KFS)  (a = KU)  (a = KUTGS)  (a = KUFS)
num2access_11
  |- r r'.
```

```
       r < 3  r' < 3  ((num2access r = num2access r')  (r = r'))
num2access_ONTO
  |- a. r. (a = num2access r)  r < 3
num2access_access2num
  |- a. num2access (access2num a) = a
num2access_thm
  |- (num2access 0 = print)  (num2access 1 = read)
     (num2access 2 = write)
num2keys_11
  |- r r'. r < 5  r' < 5  ((num2keys r = num2keys r')  (r = r'))
num2keys_ONTO
  |- a. r. (a = num2keys r)  r < 5
num2keys_keys2num
  |- a. num2keys (keys2num a) = a
num2keys_thm
  |- (num2keys 0 = KTGS)  (num2keys 1 = KFS)  (num2keys 2 = KU)
     (num2keys 3 = KUTGS)  (num2keys 4 = KUFS)
num2people_11
  |- r r'.
       r < 1  r' < 1  ((num2people r = num2people r')  (r = r'))
num2people_ONTO
  |- a. r. (a = num2people r)  r < 1
num2people_people2num
  |- a. num2people (people2num a) = a
num2people_thm
  |- num2people 0 = Ursala
num2servers_11
  |- r r'.
       r < 3
       r' < 3
       ((num2servers r = num2servers r')  (r = r'))
num2servers_ONTO
  |- a. r. (a = num2servers r)  r < 3
num2servers_servers2num
  |- a. num2servers (servers2num a) = a
num2servers_thm
  |- (num2servers 0 = AS)  (num2servers 1 = TGS)
     (num2servers 2 = FS)
people2num_11
  |- a a'. (people2num a = people2num a')  (a = a')
people2num_ONTO
  |- r. r < 1  a. r = people2num a
people2num_num2people
  |- r. r < 1  (people2num (num2people r) = r)
people2num_thm
  |- people2num Ursala = 0
people_Axiom
```

```
  |- x0. f. f Ursala = x0
people_EQ_people
  |- a a'. (a = a')  (people2num a = people2num a')
people_case_cong
  |- M M' v0.
       (M = M')  ((M' = Ursala)  (v0 = v0'))
       ((case M of Ursala => v0) = case M' of Ursala => v0')
people_case_def
  |- v0. (case Ursala of Ursala => v0) = v0
people_induction
  |- P. P Ursala  a. P a
people_nchotomy
  |- a. a = Ursala
princs_11
  |- (a a'. (PR a = PR a')  (a = a'))
     (a a'. (User a = User a')  (a = a'))
     a a'. (Key a = Key a')  (a = a')
princs_Axiom
  |- f0 f1 f2.
       fn.
         (a. fn (PR a) = f0 a)  (a. fn (User a) = f1 a)
         a. fn (Key a) = f2 a
princs_case_cong
  |- M M' f f1 f2.
       (M = M')  (a. (M' = PR a)  (f a = f' a))
       (a. (M' = User a)  (f1 a = f1' a))
       (a. (M' = Key a)  (f2 a = f2' a))
       (princs_CASE M f f1 f2 = princs_CASE M' f' f1' f2')
princs_distinct
  |- (a' a. PR a  User a')  (a' a. PR a  Key a')
     a' a. User a  Key a'
princs_induction
  |- P.
       (s. P (PR s))  (p. P (User p))  (k. P (Key k))  p. P p
princs_nchotomy
  |- pp. (s. pp = PR s)  (p. pp = User p)  k. pp = Key k
props_11
  |- (a a'. (USE a = USE a')  (a = a'))
     (a0 a1 a0' a1'.
        (PK a0 a1 = PK a0' a1')  (a0 = a0')  (a1 = a1'))
     a a'. (AC a = AC a')  (a = a')
props_Axiom
  |- f0 f1 f2.
       fn.
         (a. fn (USE a) = f0 a)
         (a0 a1. fn (PK a0 a1) = f1 a0 a1)  a. fn (AC a) = f2 a
props_case_cong
```

```
  |- M M' f f1 f2.
       (M = M')  (a. (M' = USE a)  (f a = f' a))
       (a0 a1. (M' = PK a0 a1)  (f1 a0 a1 = f1' a0 a1))
       (a. (M' = AC a)  (f2 a = f2' a))
       (props_CASE M f f1 f2 = props_CASE M' f' f1' f2')
props_distinct
  |- (a1 a0 a. USE a  PK a0 a1)  (a' a. USE a  AC a')
     a1 a0 a. PK a0 a1  AC a
props_induction
  |- P.
       (s. P (USE s))  (p k. P (PK p k))  (a. P (AC a))
       p. P p
props_nchotomy
  |- pp. (s. pp = USE s)  (p k. pp = PK p k)  a. pp = AC a
request_for_services_thm
  |- (M,Oi,Os) sat Name (Key KTGS) speaks_for Name (PR AS)
     (M,Oi,Os) sat
     prop (PK Ursala KUTGS) impf
     Name (Key KUFS) says prop (AC read)
     (M,Oi,Os) sat
     prop (PK Ursala KUTGS) impf
     Name (Key KFS) says
     Name (Key KUFS) speaks_for Name (User Ursala)
     (M,Oi,Os) sat Name (PR AS) controls prop (PK Ursala KUTGS)
     (M,Oi,Os) sat
     Name (Key KUTGS) says prop (USE (SERV FS)) andf
     Name (Key KTGS) says prop (PK Ursala KUTGS)
     (M,Oi,Os) sat
     Name (Key KFS) says
     Name (Key KUFS) speaks_for Name (User Ursala) andf
     Name (Key KUFS) says prop (AC read)
servers2num_11
  |- a a'. (servers2num a = servers2num a')  (a = a')
servers2num_ONTO
  |- r. r < 3  a. r = servers2num a
servers2num_num2servers
  |- r. r < 3  (servers2num (num2servers r) = r)
servers2num_thm
  |- (servers2num AS = 0)  (servers2num TGS = 1)
     (servers2num FS = 2)
servers_Axiom
  |- x0 x1 x2. f. (f AS = x0)  (f TGS = x1)  (f FS = x2)
servers_EQ_servers
  |- a a'. (a = a')  (servers2num a = servers2num a')
servers_case_cong
  |- M M' v0 v1 v2.
       (M = M')  ((M' = AS)  (v0 = v0'))
```

```
       ((M' = TGS)  (v1 = v1'))  ((M' = FS)  (v2 = v2'))
       ((case M of AS => v0 | TGS => v1 | FS => v2) =
        case M' of AS => v0' | TGS => v1' | FS => v2')
servers_case_def
  |- (v0 v1 v2.
        (case AS of AS => v0 | TGS => v1 | FS => v2) = v0)
     (v0 v1 v2.
        (case TGS of AS => v0 | TGS => v1 | FS => v2) = v1)
     v0 v1 v2. (case FS of AS => v0 | TGS => v1 | FS => v2) = v2
servers_distinct
  |- AS  TGS  AS  FS  TGS  FS
servers_induction
  |- P. P AS  P FS  P TGS  a. P a
servers_nchotomy
  |- a. (a = AS)  (a = TGS)  (a = FS)
serviceKey_11
  |- (a a'. (SERV a = SERV a')  (a = a'))
     a a'. (K a = K a')  (a = a')
serviceKey_Axiom
  |- f0 f1. fn. (a. fn (SERV a) = f0 a)  a. fn (K a) = f1 a
serviceKey_case_cong
  |- M M' f f1.
       (M = M')  (a. (M' = SERV a)  (f a = f' a))
       (a. (M' = K a)  (f1 a = f1' a))
       (serviceKey_CASE M f f1 = serviceKey_CASE M' f' f1')
serviceKey_distinct
  |- a' a. SERV a  K a'
serviceKey_induction
  |- P. (s. P (SERV s))  (k. P (K k))  s. P s
serviceKey_nchotomy
  |- ss. (s. ss = SERV s)  k. ss = K k
service_request_thm
  |- (M,Oi,Os) sat Name (Key KFS) speaks_for Name (PR TGS)
     (M,Oi,Os) sat Name (User Ursala) controls prop (AC print)
     (M,Oi,Os) sat
     Name (PR TGS) controls
     Name (Key KUFS) speaks_for Name (User Ursala)
     (M,Oi,Os) sat
     Name (Key KFS) says
     Name (Key KUFS) speaks_for Name (User Ursala) andf
     Name (Key KUFS) says prop (AC print)
     (M,Oi,Os) sat prop (AC print)
session_key_receipt_thm
  |- (M,Oi,Os) sat Name (Key KU) speaks_for Name (PR AS)
     (M,Oi,Os) sat
     prop (USE (K KUTGS)) impf
     Name (Key KUTGS) says prop (USE (SERV FS))
```

```
         (M,Oi,Os) sat Name (PR AS) controls prop (USE (K KUTGS))
         (M,Oi,Os) sat
         Name (Key KU) says prop (USE (K KUTGS)) andf
         Name (Key KTGS) says prop (PK Ursala KUTGS)
         (M,Oi,Os) sat
         Name (Key KUTGS) says prop (USE (SERV FS)) andf
         Name (Key KTGS) says prop (PK Ursala KUTGS)
Exporting theory "exam4" ... done.
Theory "exam4" took 1.2s to build
structure exam4Script:
  sig
  end
val it = (): unit
>

Process HOL finished
```

3 Question 2

3.1 Problem Statement

This problem explores defining a simple state machine in the form of simpleOpener Theory. The dataytpes are defined below as well as the next state, next output, and other relevant theorems for the state machine. Source for the state machine can be found in appendix B.

3.2 Datatypes/Theorems

command = i0 | i1

output = o0 | o1

state = S0 | S1

$\vdash$ (simpleOpenerns S0 i1 = S1) $\wedge$ (simpleOpenerns S1 i0 = S0)

$\vdash$ (simpleOpenerout S0 i1 = o1) $\wedge$ (simpleOpenerout S1 i0 = o0)

$\vdash$ i0 $\neq$ i1

$\vdash$ o0 $\neq$ o1

$\vdash$ ($\forall$ *ins outs*.
TR i1 (CFG (i1::*ins*) S0 *outs*) (CFG *ins* S1 (o1::*outs*))) $\wedge$
$\forall$ *ins outs*.
TR i0 (CFG (i0::*ins*) S1 *outs*) (CFG *ins* S0 (o0::*outs*))

$\vdash \forall P.\ P$ S0 i1 $\wedge\ P$ S1 i0 $\wedge\ P$ S0 i0 $\wedge\ P$ S1 i1 $\Rightarrow \forall v\ v_1.\ P\ v\ v_1$

$\vdash \forall P.\ P$ S0 i1 $\wedge\ P$ S1 i0 $\wedge\ P$ S0 i0 $\wedge\ P$ S1 i1 $\Rightarrow \forall v\ v_1.\ P\ v\ v_1$

$\vdash$ ($\forall x\ x1s\ s_1\ out1s\ x2s\ out2s\ s_2$.
TR x (CFG *x1s* s_1 *out1s*) (CFG *x2s* s_2 *out2s*) $\Longleftrightarrow$
$\exists$ *NS Out ins*.
(*x1s* = x::*ins*) $\wedge$ (*x2s* = *ins*) $\wedge$ (s_2 = *NS* s_1 x) $\wedge$
(*out2s* = *Out* s_1 x::*out1s*)) $\wedge$
$\forall x\ x1s\ s_1\ out1s\ x2s\ out2s$.
TR x (CFG *x1s* s_1 *out1s*)
(CFG *x2s* (simpleOpenerns s_1 x)
(simpleOpenerout s_1 x::*out2s*)) $\Longleftrightarrow$
$\exists$ *ins*. (*x1s* = x::*ins*) $\wedge$ (*x2s* = *ins*) $\wedge$ (*out2s* = *out1s*)

$\vdash \forall s\ x\ ins\ outs$.
TR x (CFG (x::*ins*) s *outs*)
(CFG *ins* (simpleOpenerns s x)
(simpleOpenerout s x::*outs*))

$\vdash$ TR x (CFG (x::*ins*) s *outs*)
(CFG *ins* (simpleOpenerns s x)
(simpleOpenerout s x::*outs*)) $\Longleftrightarrow$
Trans x s (simpleOpenerns s x)

$\vdash$ S0 $\neq$ S1

3.3 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]

> > > > > > > > <<HOL message: Created theory "simpleOpener">>
<<HOL message: Defined type: "command">>
<<HOL message: Defined type: "state">>
<<HOL message: Defined type: "output">>
<<HOL message: mk_functional:
  pattern completion has added 2 clauses to the original specification.>>
<<HOL warning: GrammarDeltas.revise_data:
  Grammar-deltas:
    overload_on("simpleOpenerns_tupled")
  invalidated by DelConstant(simpleOpener$simpleOpenerns_tupled)>>
Equations stored under "simpleOpenerns_def".
Induction stored under "simpleOpenerns_ind".
<<HOL message: mk_functional:
  pattern completion has added 2 clauses to the original specification.>>
<<HOL warning: GrammarDeltas.revise_data:
  Grammar-deltas:
    overload_on("simpleOpenerout_tupled")
  invalidated by DelConstant(simpleOpener$simpleOpenerout_tupled)>>
Equations stored under "simpleOpenerout_def".
Induction stored under "simpleOpenerout_ind".
Exporting theory "simpleOpener" ... done.
Theory "simpleOpener" took 0.31254s to build
Theory: simpleOpener

Parents:
    sm

Type constants:
    command 0
    output 0
    state 0

Term constants:
    S0                :state
    S1                :state
    command2num       :command -> num
    command_CASE      :command ->  ->  ->
```

```
    command_size      :command -> num
    i0                :command
    i1                :command
    num2command       :num -> command
    num2output        :num -> output
    num2state         :num -> state
    o0                :output
    o1                :output
    output2num        :output -> num
    output_CASE       :output ->  ->  ->
    output_size       :output -> num
    simpleOpenerns    :state -> command -> state
    simpleOpenerout   :state -> command -> output
    state2num         :state -> num
    state_CASE        :state ->  ->  ->
    state_size        :state -> num

Definitions:
    @tempS0_def
      |- S0 = num2state 0
    @tempS1_def
      |- S1 = num2state 1
    @tempi0_def
      |- i0 = num2command 0
    @tempi1_def
      |- i1 = num2command 1
    @tempo0_def
      |- o0 = num2output 0
    @tempo1_def
      |- o1 = num2output 1
    command_BIJ
      |- (a. num2command (command2num a) = a)
         r. (n. n < 2) r  (command2num (num2command r) = r)
    command_CASE
      |- x v0 v1.
           (case x of i0 => v0 | i1 => v1) =
           (m. if m = 0 then v0 else v1) (command2num x)
    command_TY_DEF
      |- rep. TYPE_DEFINITION (n. n < 2) rep
    command_size_def
      |- x. command_size x = 0
    output_BIJ
      |- (a. num2output (output2num a) = a)
         r. (n. n < 2) r  (output2num (num2output r) = r)
    output_CASE
      |- x v0 v1.
           (case x of o0 => v0 | o1 => v1) =
```

```
        (m. if m = 0 then v0 else v1) (output2num x)
    output_TY_DEF
      |- rep. TYPE_DEFINITION (n. n < 2) rep
    output_size_def
      |- x. output_size x = 0
    state_BIJ
      |- (a. num2state (state2num a) = a)
         r. (n. n < 2) r  (state2num (num2state r) = r)
    state_CASE
      |- x v0 v1.
           (case x of S0 => v0 | S1 => v1) =
           (m. if m = 0 then v0 else v1) (state2num x)
    state_TY_DEF
      |- rep. TYPE_DEFINITION (n. n < 2) rep
    state_size_def
      |- x. state_size x = 0

Theorems:
    command2num_11
      |- a a'. (command2num a = command2num a')  (a = a')
    command2num_ONTO
      |- r. r < 2  a. r = command2num a
    command2num_num2command
      |- r. r < 2  (command2num (num2command r) = r)
    command2num_thm
      |- (command2num i0 = 0)  (command2num i1 = 1)
    command_Axiom
      |- x0 x1. f. (f i0 = x0)  (f i1 = x1)
    command_EQ_command
      |- a a'. (a = a')  (command2num a = command2num a')
    command_case_cong
      |- M M' v0 v1.
           (M = M')  ((M' = i0)  (v0 = v0'))
           ((M' = i1)  (v1 = v1'))
           ((case M of i0 => v0 | i1 => v1) =
            case M' of i0 => v0' | i1 => v1')
    command_case_def
      |- (v0 v1. (case i0 of i0 => v0 | i1 => v1) = v0)
         v0 v1. (case i1 of i0 => v0 | i1 => v1) = v1
    command_distinct
      |- i0  i1
    command_distinct_clauses
      |- i0  i1
    command_induction
      |- P. P i0  P i1  a. P a
    command_nchotomy
      |- a. (a = i0)  (a = i1)
```

```
datatype_command
  |- DATATYPE (command i0 i1)
datatype_output
  |- DATATYPE (output o0 o1)
datatype_state
  |- DATATYPE (state S0 S1)
num2command_11
  |- r r'.
       r < 2
       r' < 2
       ((num2command r = num2command r')  (r = r'))
num2command_ONTO
  |- a. r. (a = num2command r)  r < 2
num2command_command2num
  |- a. num2command (command2num a) = a
num2command_thm
  |- (num2command 0 = i0)  (num2command 1 = i1)
num2output_11
  |- r r'.
       r < 2  r' < 2  ((num2output r = num2output r')  (r = r'))
num2output_ONTO
  |- a. r. (a = num2output r)  r < 2
num2output_output2num
  |- a. num2output (output2num a) = a
num2output_thm
  |- (num2output 0 = o0)  (num2output 1 = o1)
num2state_11
  |- r r'.
       r < 2  r' < 2  ((num2state r = num2state r')  (r = r'))
num2state_ONTO
  |- a. r. (a = num2state r)  r < 2
num2state_state2num
  |- a. num2state (state2num a) = a
num2state_thm
  |- (num2state 0 = S0)  (num2state 1 = S1)
output2num_11
  |- a a'. (output2num a = output2num a')  (a = a')
output2num_ONTO
  |- r. r < 2  a. r = output2num a
output2num_num2output
  |- r. r < 2  (output2num (num2output r) = r)
output2num_thm
  |- (output2num o0 = 0)  (output2num o1 = 1)
output_Axiom
  |- x0 x1. f. (f o0 = x0)  (f o1 = x1)
output_EQ_output
  |- a a'. (a = a')  (output2num a = output2num a')
```

```
output_case_cong
  |- M M' v0 v1.
       (M = M')  ((M' = o0)  (v0 = v0'))
       ((M' = o1)  (v1 = v1'))
       ((case M of o0 => v0 | o1 => v1) =
        case M' of o0 => v0' | o1 => v1')
output_case_def
  |- (v0 v1. (case o0 of o0 => v0 | o1 => v1) = v0)
     v0 v1. (case o1 of o0 => v0 | o1 => v1) = v1
output_distinct
  |- o0  o1
output_distinct_clauses
  |- o0  o1
output_induction
  |- P. P o0  P o1  a. P a
output_nchotomy
  |- a. (a = o0)  (a = o1)
simpleCounter_rules
  |- (ins outs.
        TR i1 (CFG (i1::ins) S0 outs) (CFG ins S1 (o1::outs)))
     ins outs.
       TR i0 (CFG (i0::ins) S1 outs) (CFG ins S0 (o0::outs))
simpleOpenerTR_clauses
  |- (x x1s s1 out1s x2s out2s s2.
        TR x (CFG x1s s1 out1s) (CFG x2s s2 out2s)
        NS Out ins.
          (x1s = x::ins)  (x2s = ins)  (s2 = NS s1 x)
          (out2s = Out s1 x::out1s))
     x x1s s1 out1s x2s out2s.
       TR x (CFG x1s s1 out1s)
         (CFG x2s (simpleOpenerns s1 x)
            (simpleOpenerout s1 x::out2s))
       ins. (x1s = x::ins)  (x2s = ins)  (out2s = out1s)
simpleOpenerTR_rules
  |- s x ins outs.
       TR x (CFG (x::ins) s outs)
         (CFG ins (simpleOpenerns s x) (simpleOpenerout s x::outs))
simpleOpenerTrans_Equiv_TR
  |- TR x (CFG (x::ins) s outs)
       (CFG ins (simpleOpenerns s x) (simpleOpenerout s x::outs))
     Trans x s (simpleOpenerns s x)
simpleOpenerns_def
  |- (simpleOpenerns S0 i1 = S1)  (simpleOpenerns S1 i0 = S0)
simpleOpenerns_ind
  |- P. P S0 i1  P S1 i0  P S0 i0  P S1 i1  v v1. P v v1
simpleOpenerout_def
  |- (simpleOpenerout S0 i1 = o1)  (simpleOpenerout S1 i0 = o0)
```

```
    simpleOpenerout_ind
      |- P. P S0 i1  P S1 i0  P S0 i0  P S1 i1  v v1. P v v1
    state2num_11
      |- a a'. (state2num a = state2num a')  (a = a')
    state2num_ONTO
      |- r. r < 2  a. r = state2num a
    state2num_num2state
      |- r. r < 2  (state2num (num2state r) = r)
    state2num_thm
      |- (state2num S0 = 0)  (state2num S1 = 1)
    state_Axiom
      |- x0 x1. f. (f S0 = x0)  (f S1 = x1)
    state_EQ_state
      |- a a'. (a = a')  (state2num a = state2num a')
    state_case_cong
      |- M M' v0 v1.
           (M = M')  ((M' = S0)  (v0 = v0'))
           ((M' = S1)  (v1 = v1'))
           ((case M of S0 => v0 | S1 => v1) =
            case M' of S0 => v0' | S1 => v1')
    state_case_def
      |- (v0 v1. (case S0 of S0 => v0 | S1 => v1) = v0)
         v0 v1. (case S1 of S0 => v0 | S1 => v1) = v1
    state_distinct
      |- S0  S1
    state_distinct_clauses
      |- S0  S1
    state_induction
      |- P. P S0  P S1  a. P a
    state_nchotomy
      |- a. (a = S0)  (a = S1)
structure simpleOpenerScript:
  sig

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

>
```

4 Question 3

In the healthcare industry, access control polocies can be overuled in cases of emergency in order to swiftly get patients the care that they need. Unfortunately, this also opens the door to people who want to take advantage of these "break the glass" policies. The policy spaces described in the paper may offer a good way to provide a methodology to deciding when to break the polocies. The paper describes a good way of splitting up the polocies that should never be broken into their own category, which is good in keeping malicious people from resources that should never be exploited. The more interesting cases are the policis in the unknown space. The enviornmental effects that govern these polocies may help healthcare providers react to disastrous cases are situations more quickly, since they have a methodology in breaking their poloices instead of leaving them up to interpretation by individual gatekeepers. On the flip side, the complications added to the access control logic to incorporate this dynamic behavior may perhaps lead to more unforseen vulnerabilities, as the environment could behave in unpredictable ways. Therefore, much care and effort must be made when decision these policy spaces.

A Source For exam4Script.sml

The following code is from *HOL/exam4Script.sml*

```
(*****************************************************************************)
(*  Exam4 Theory: Access Control Mechanisms
*)
(*  Author: Alfred Murabito
*)
(*  Date: 3/26/2020
*)
(*****************************************************************************)

structure exam4Script = struct

(* only necessary when working interactively
app load ["acl_infRules","aclrulesTheory","aclDrulesTheory","exam4Theory"]
open acl_infRules aclrulesTheory aclDrulesTheory exam4Theory
*)

open HolKernel boolLib Parse bossLib
open acl_infRules aclrulesTheory aclDrulesTheory

val _ = new_theory "exam4"

(* Theory Definitions *)
val _ =
Datatype
`servers = AS | TGS | FS`

val _ =
Datatype
`people = Ursala`

val _ =
Datatype
`keys = KTGS | KFS | KU | KUTGS | KUFS`

val _ =
Datatype
`princs = PR servers | User people | Key keys`

val _ =
Datatype
`access = print | read | write`

val _ =
Datatype
```

```
‘serviceKey = SERV servers | K keys‘

val _ =
Datatype
‘props = USE serviceKey | PK people keys | AC access‘


val forma1 = ‘‘Name (User Ursala) says prop (USE (SERV TGS)):(props,princs,'d,'e
val forma2 = ‘‘Name (User Ursala) controls prop (USE (SERV TGS)):(props,princs,'
val forma3 = ‘‘prop (USE (SERV TGS)) impf (Name (Key KU) says prop (USE (K KUTGS
                                                                        andf
val [forma1_thm,forma2_thm,forma3_thm] = map ACL_ASSUM [forma1,forma2,forma3]

val forma4_thm = CONTROLS forma2_thm forma1_thm
val forma_thm = ACL_MP forma4_thm forma3_thm
val init_auth_thm = DISCH_ALL forma_thm

val _ = save_thm("init_auth_thm",init_auth_thm)


val formb1 = ‘‘(Name (Key KU) says prop (USE (K KUTGS)) andf
               Name (Key KTGS) says prop (PK Ursala KUTGS)):(props,princs,'d,'e)

val formb2 = ‘‘Name (PR AS) controls prop (USE (K KUTGS)):(props,princs,'d,'e)For
val formb3 = ‘‘(Name (Key KU) speaks_for Name (PR AS)):(props,princs,'d,'e)Form‘
val formb4 = ‘‘prop (USE (K KUTGS)) impf Name (Key KUTGS) says prop (USE (SERV FS

val [formb1_thm,formb2_thm,formb3_thm,formb4_thm] = map ACL_ASSUM[formb1,formb2,
val formb5_thm = ACL_SIMP1 formb1_thm
val formb6_thm = ACL_SIMP2 formb1_thm
val formb7_thm = SPEAKS_FOR formb3_thm formb5_thm
val formb8_thm = CONTROLS formb2_thm formb7_thm
val formb9_thm = ACL_MP formb8_thm formb4_thm
val formb_thm = ACL_CONJ formb9_thm formb6_thm
val session_key_receipt_thm = DISCH_ALL formb_thm

val _ = save_thm("session_key_receipt_thm",session_key_receipt_thm)


val formc1 = ‘‘Name (Key KUTGS) says prop (USE (SERV FS)) andf
               Name (Key KTGS) says prop (PK Ursala KUTGS):(props,princs,'d,'e)Fo

val formc2 = ‘‘(Name (PR AS)) controls (prop (PK Ursala KUTGS)):(props,princs,'d
val formc3 = ‘‘(Name (Key KTGS) speaks_for (Name (PR AS))):(props,princs,'d,'e)Fo

val formc4 = ‘‘prop (PK Ursala KUTGS) impf (Name (Key KFS) says ((Name (Key KUFS
val formc5 = ‘‘prop (PK Ursala KUTGS) impf (Name (Key KUFS) says prop (AC read))
```

```
val [formc1_thm,formc2_thm,formc3_thm,formc4_thm,formc5_thm] = map ACL_ASSUM[formc1,f

val formc6_thm = ACL_SIMP1 formc1_thm
val formc7_thm = ACL_SIMP2 formc1_thm
val formc8_thm = SPEAKS_FOR formc3_thm formc7_thm
val formc9_thm = CONTROLS formc2_thm formc8_thm
val formc10_thm = ACL_MP formc9_thm formc4_thm
val formc11_thm = ACL_MP formc9_thm formc5_thm
val formc_thm = ACL_CONJ formc10_thm formc11_thm
val request_for_services_thm = DISCH_ALL formc_thm

val _ = save_thm("request_for_services_thm",request_for_services_thm)


val formd1 = ``(Name (Key KFS) says (Name (Key KUFS) speaks_for Name (User Ursala)))
                              (Name (Key KUFS) says prop (AC print)):(props,princs,'d,'e
val formd2 = ``(Name (Key KFS) speaks_for Name (PR TGS)):(props,princs,'d,'e)Form``
val formd3 = ``(Name (PR TGS) controls (Name (Key KUFS) speaks_for Name (User Ursala)
val formd4 = ``(Name (User Ursala) controls prop(AC print)):(props,princs,'d,'e)Form`

val [formd1_thm,formd2_thm,formd3_thm,formd4_thm] = map ACL_ASSUM[formd1,formd2,formd
val formd5_thm = ACL_SIMP1 formd1_thm
val formd6_thm = ACL_SIMP2 formd1_thm
val formd7_thm = SPEAKS_FOR formd2_thm formd5_thm
val formd8_thm = CONTROLS formd3_thm formd7_thm
val formd9_thm = SPEAKS_FOR formd8_thm formd6_thm
val formd_thm = CONTROLS formd4_thm formd9_thm
val service_request_thm = DISCH_ALL formd_thm

val _ = save_thm("service_request_thm",service_request_thm)

val _ = print_theory "-";

val _ = export_theory();

end
```

B Source For simpleOpenerScript.sml

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

```
(*****************************************************************************)
(* Exam 4: Simple Opener Theory
*)
(* Alfred Murabito
*)
```

```
(* March 26 2020
*)
(*****************************************************************************)

structure simpleOpenerScript = struct

(* interactive mode
app load ["TypeBase","smTheory","sminfRules","simpleOpenerTheory"];
open TypeBase smTheory sminfRules simpleOpenerTheory
*)

open HolKernel boolLib Parse bossLib
open TypeBase smTheory sminfRules

val _ = new_theory "simpleOpener"

(* ------------------------------------------------------------------------- *)
(* Define the inputs
*)
(* ------------------------------------------------------------------------- *)
val _ =
Datatype
`command = i0 | i1 `

val command_distinct_clauses = distinct_of ``:command``
val _ = save_thm("command_distinct_clauses",command_distinct_clauses)

(* ------------------------------------------------------------------------- *)
(* Define the states
*)
(* ------------------------------------------------------------------------- *)
val _ =
Datatype
`state = S0 | S1`

val state_distinct_clauses = distinct_of ``:state``
val _ = save_thm("state_distinct_clauses",state_distinct_clauses)

(* ------------------------------------------------------------------------- *)
(* Define the outputs
*)
(* ------------------------------------------------------------------------- *)
val _ =
Datatype
`output = o0 | o1`

val output_distinct_clauses = distinct_of ``:output``
```

```
val _ = save_thm("output_distinct_clauses",output_distinct_clauses)

(* ------------------------------------------------------------------------- *)
(* Define next-state function for machine simpleOpener                       *)
(* ------------------------------------------------------------------------- *)
val simpleOpenerns_def =
Define`(simpleOpenerns S0 i1 = S1) /\ (simpleOpenerns S1 i0 = S0)`

(* ------------------------------------------------------------------------- *)
(* Define next-output function for machine simpleOpener                      *)
(* ------------------------------------------------------------------------- *)
val simpleOpenerout_def =
Define`(simpleOpenerout S0 i1 = o1) /\ (simpleOpenerout S1 i0 = o0)`

(* ------------------------------------------------------------------------- *)
(* Specialize TR_rules to simpleOpener                                       *)
(* ------------------------------------------------------------------------- *)
val simpleOpenerTR_rules = SPEC_TR ``simpleOpenerns`` ``simpleOpenerout``

val _ = save_thm("simpleOpenerTR_rules",simpleOpenerTR_rules)

(* ------------------------------------------------------------------------- *)
(* Specialize TR_clauses to simpleOpener                                     *)
(* ------------------------------------------------------------------------- *)
val simpleOpenerTR_clauses = SPEC_TR_clauses ``simpleOpenerns`` ``simpleOpenerout``

val _ = save_thm("simpleOpenerTR_clauses",simpleOpenerTR_clauses)

(* ------------------------------------------------------------------------- *)
(* Specialized Trans_Equiv_TR theorem to simpleOpener                        *)
(* ------------------------------------------------------------------------- *)
val simpleOpenerTrans_Equiv_TR = SPEC_Trans_Equiv_TR ``simpleOpenerns`` ``simpleOpene

val _ = save_thm("simpleOpenerTrans_Equiv_TR",simpleOpenerTrans_Equiv_TR)

(* ------------------------------------------------------------------------- *)
(* Theorems corresponding to the tabular specification of simpleOpener       *)
(* ------------------------------------------------------------------------- *)
val th1 = REWRITE_RULE[simpleOpenerns_def,simpleOpenerout_def](SPECL[``S0``,``i1``] s
val th2 = REWRITE_RULE[simpleOpenerns_def,simpleOpenerout_def](SPECL[``S1``,``i0``] s
```

```
val simpleCounter_rules = LIST_CONJ [th1,th2]

val _ = save_thm("simpleCounter_rules",simpleCounter_rules)

val _ = export_theory ()
val _ = print_theory "-"

end
```